Task 3 - 2D computer vision - Darts game analysis - Report

Your task is to analyze the images of the darts game, find the score and the winner (red or green) with the methods of computer vision. You can use algorithmic, deep learning approaches or their mixture, just show your skills.

Deliverable:

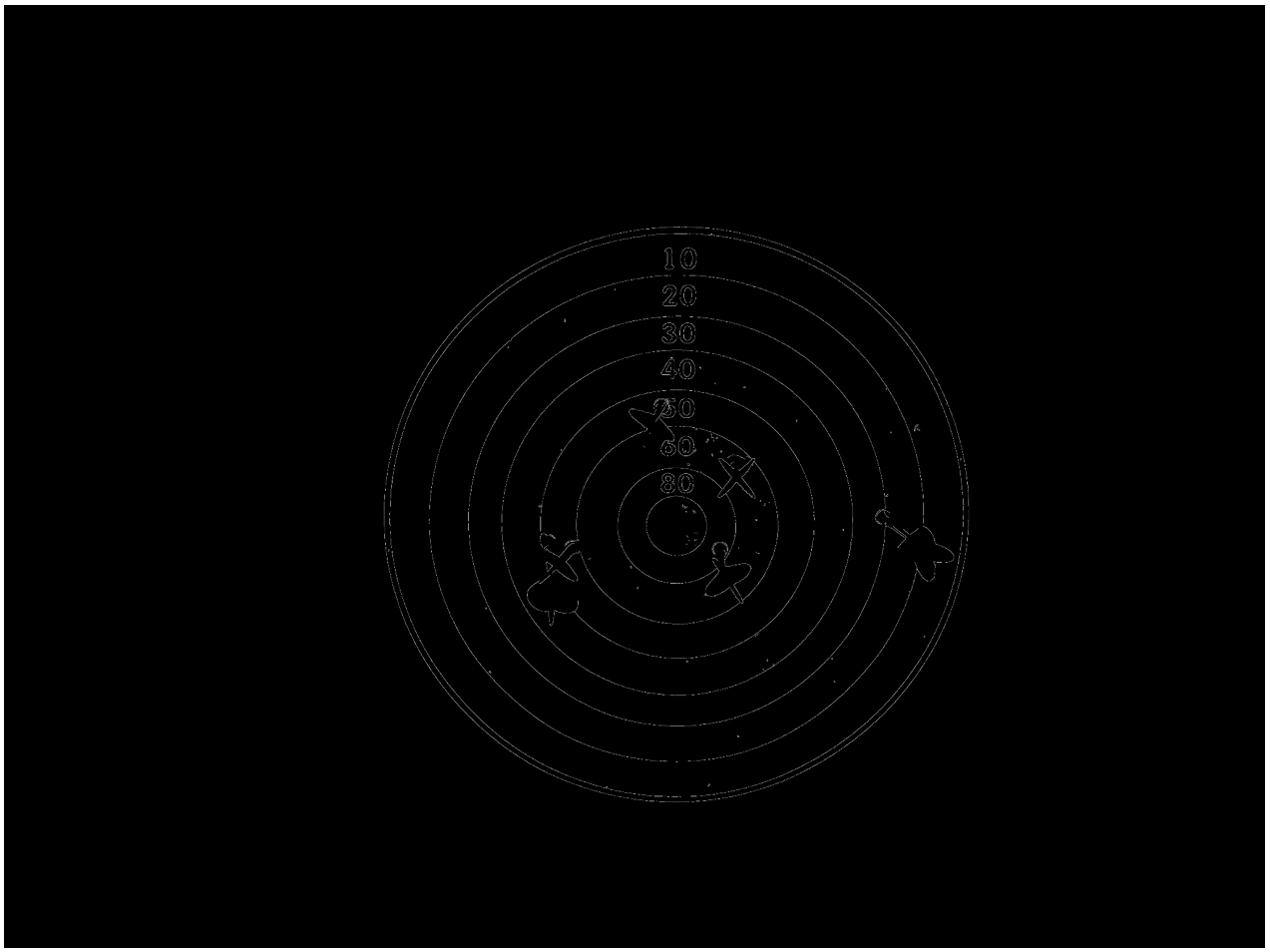
- Working code (it should be possible to install all the necessary packages to run it from *requirements.txt* using *pip*),
- A report [in English] on the principle of operation of your solution (better with illustrations and explanations of the chosen solution), as well as discussion of the results obtained.
- The initial game images with found scores.

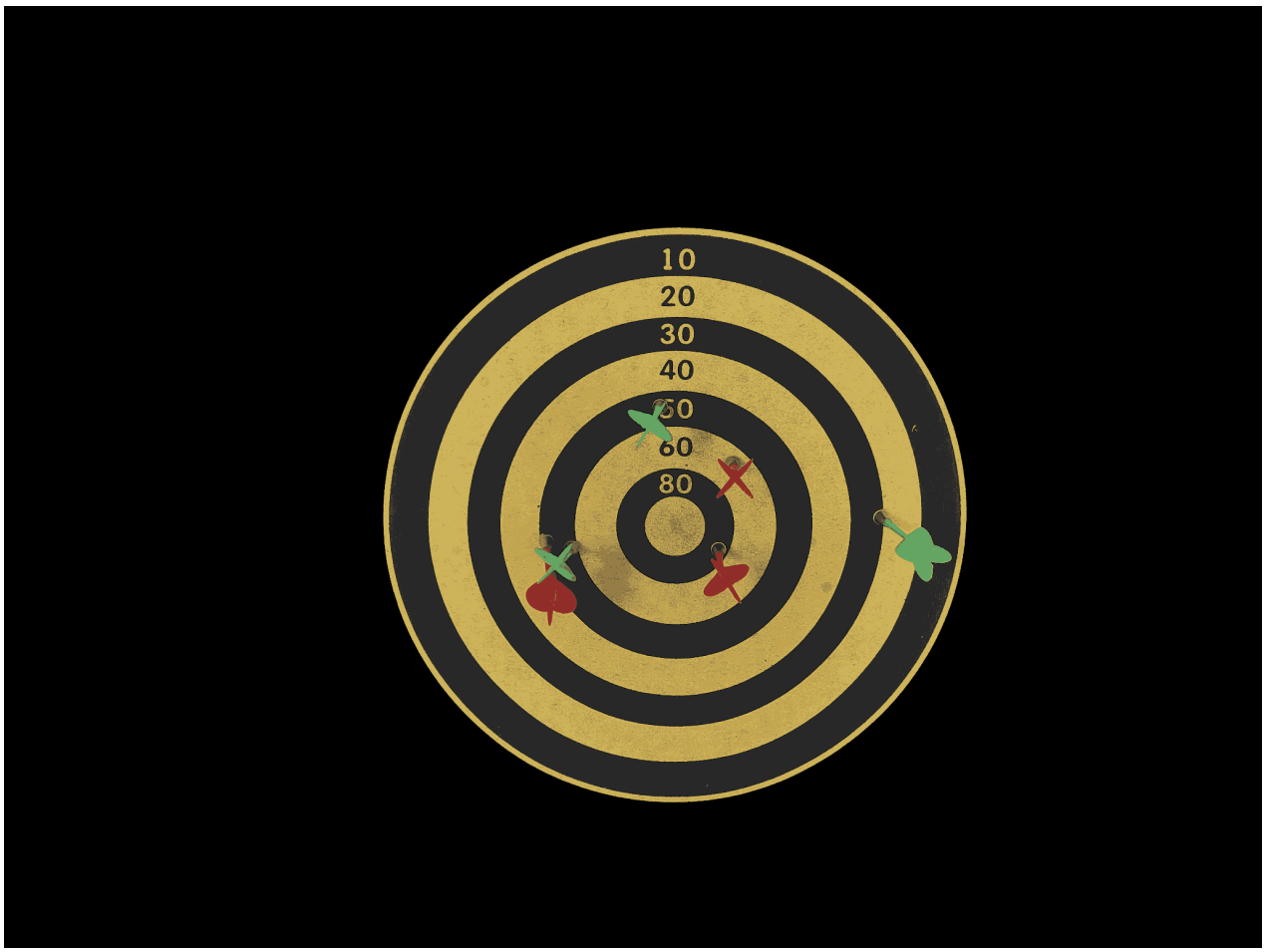
Introduction

In this task, I introduce an algorithm crafted to detect darts on a dartboard, particularly focusing on images captured by a phone camera. Throughout the development, I experimented with several approaches to optimize the detection process, which I will elaborate on in detail below. Initially, I will provide a general overview of the algorithm's development.

General Pipeline

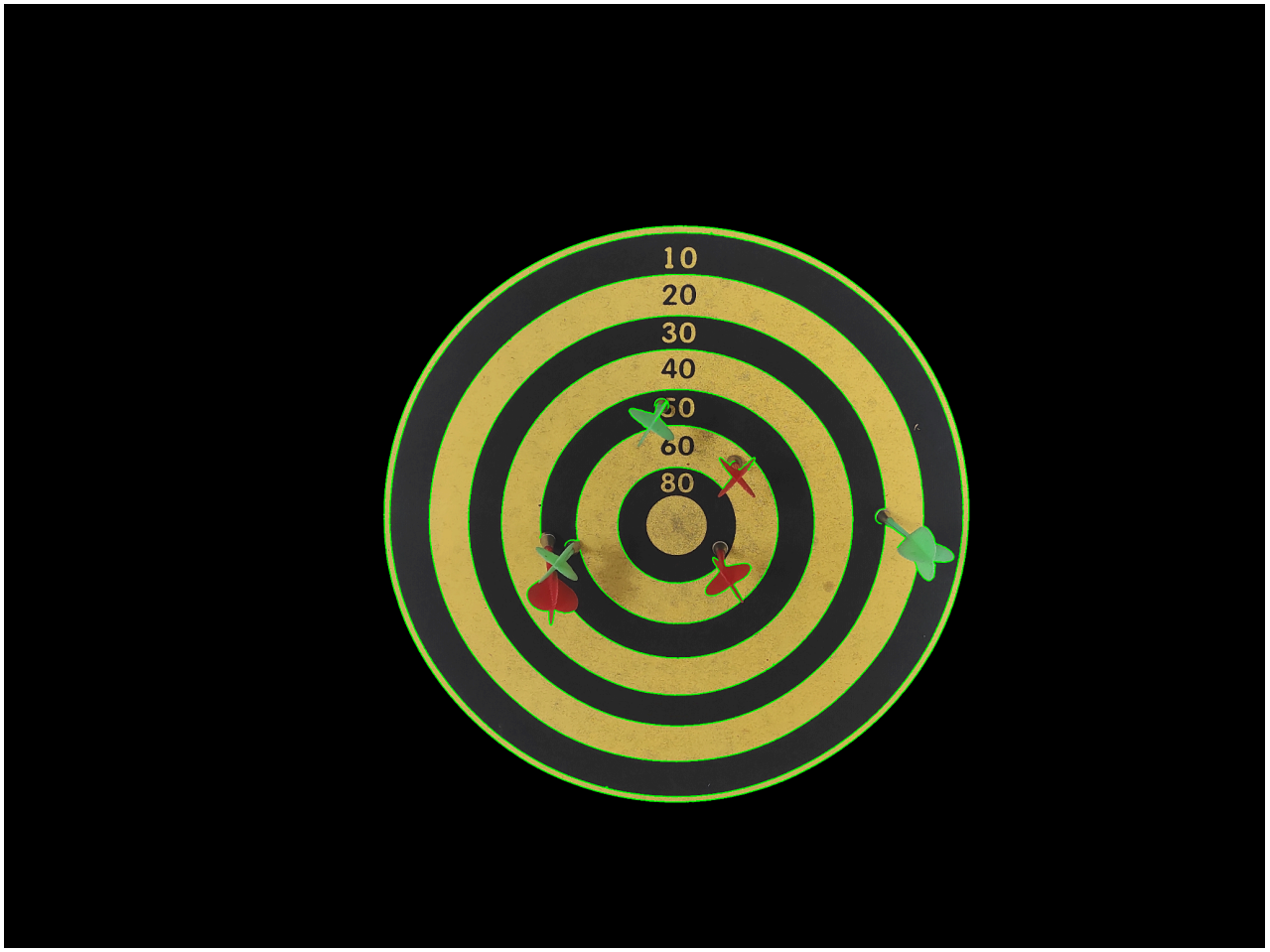
1. **Detecting the Largest External Circle**:
    - The algorithm begins by identifying the largest external circle, which corresponds to the contour of the dart





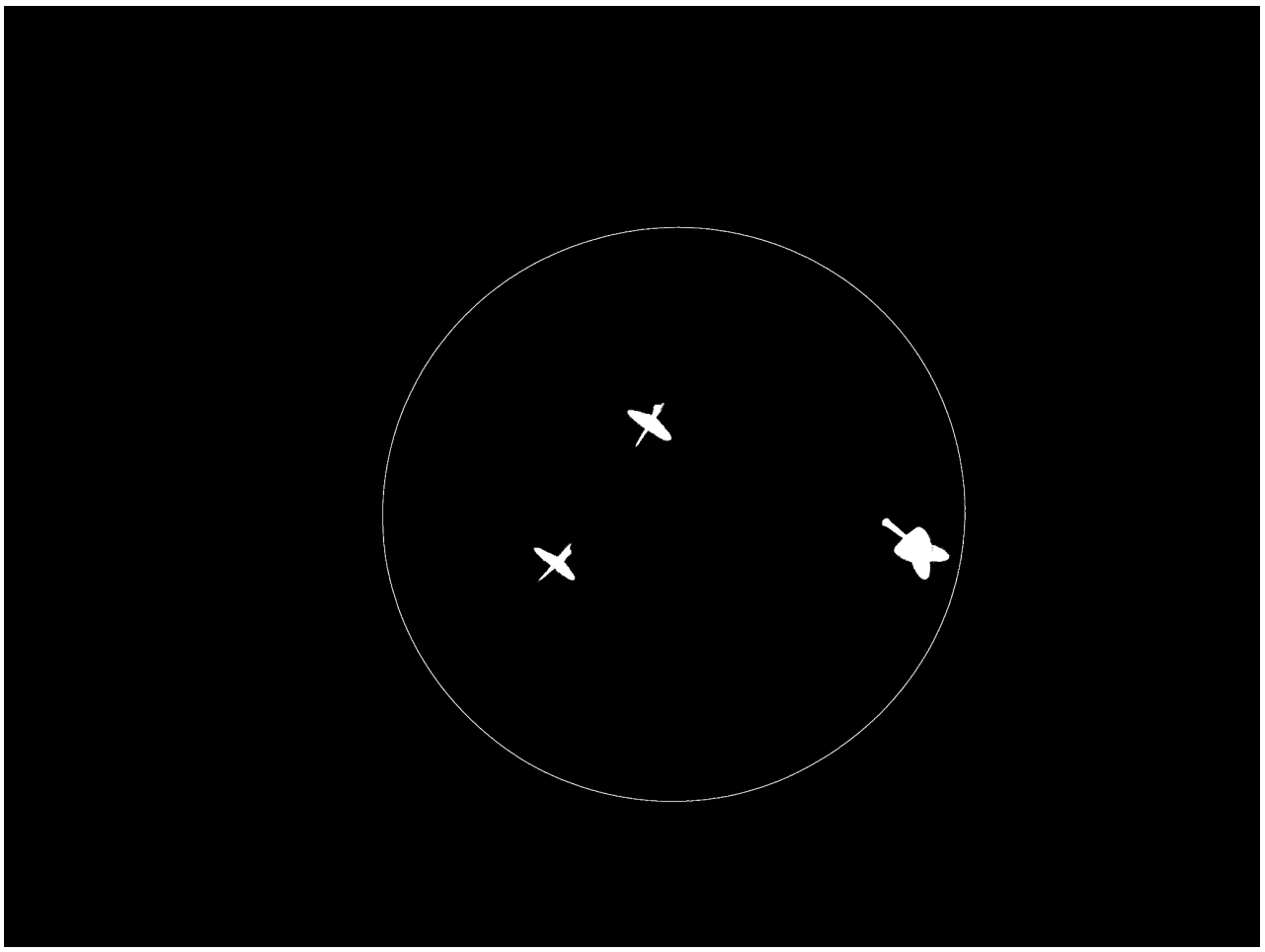
2. **Locating the Center of the Dartboard**:
    - Next, it calculates the precise center of the dartboard, which serves as a critical reference point for further processing.

3. **Identifying Internal Contours**:
    - The algorithm then detects internal contours to segment the areas between neighboring circles, which is essential for accurate dart placement analysis.

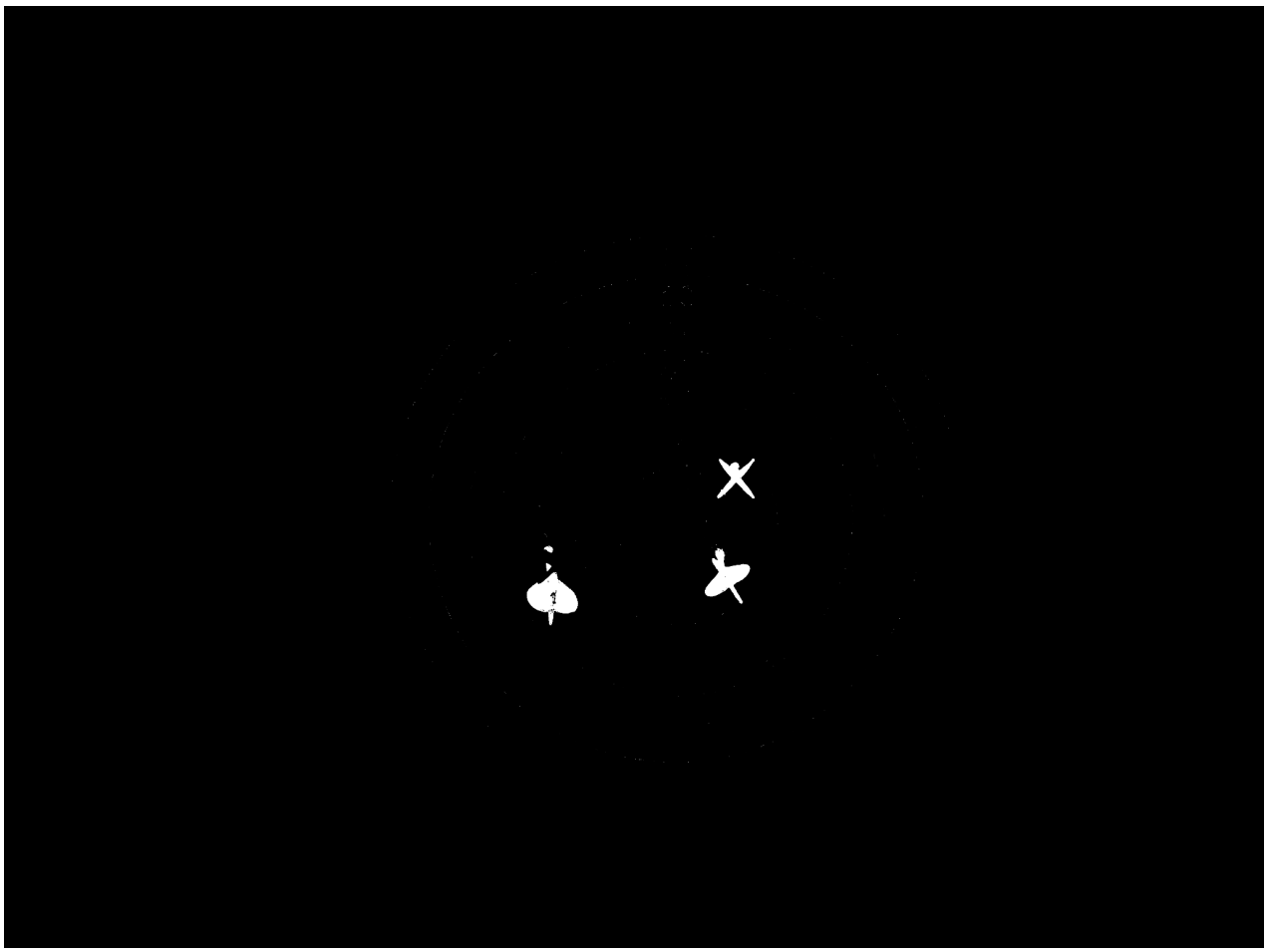


4. **Color Segmentation of Darts**:
    - Darts are segmented based on the color of their tips, specifically targeting red, green, and gold colors

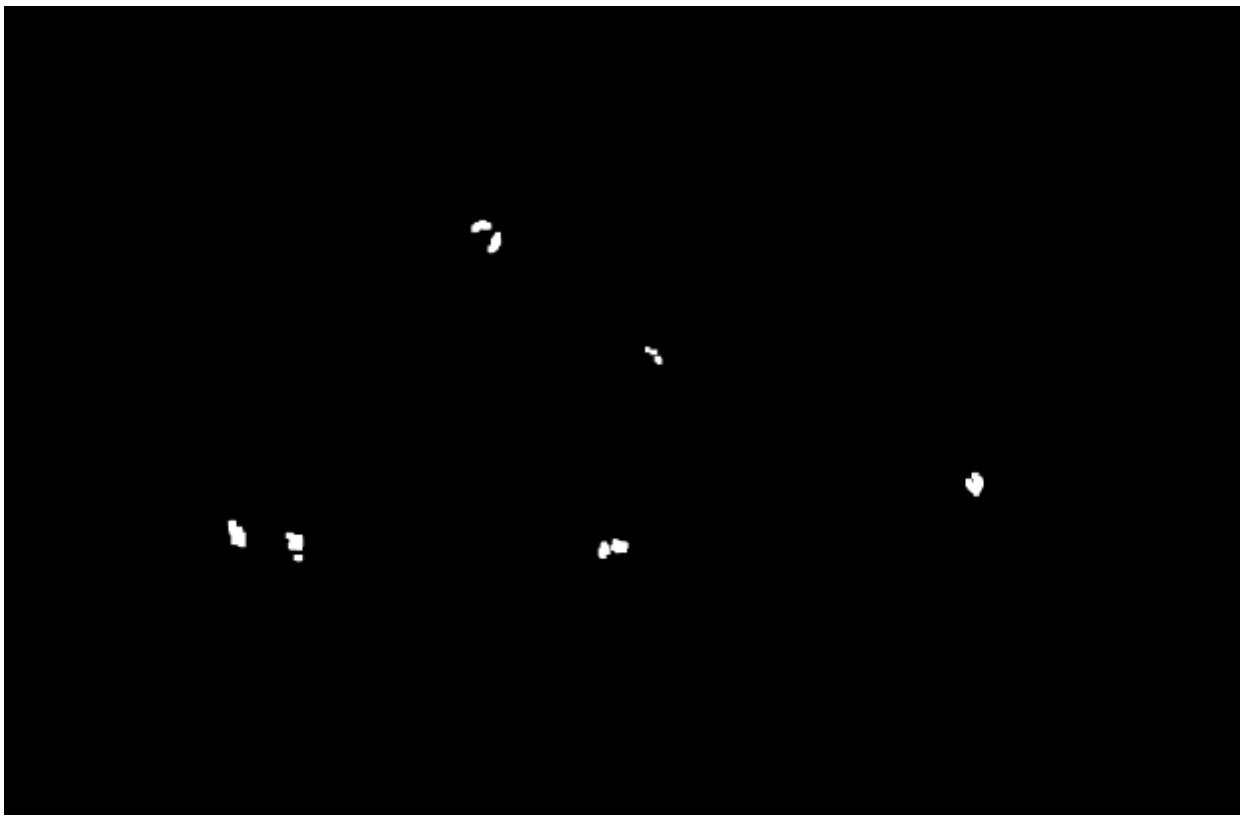
Green:



Red:



5. **Masking Non-Gold Pixels**:
    - The process involves masking out all pixels except for the gold ones, allowing for the isolation of dart tips.



6. **Clustering Dart Tips**:
    - The algorithm identifies and clusters the gold pixels, pinpointing the centers of each cluster to locate individual dart tips (typically resulting in six clusters for six tips).

7. **Calculating Radii for Internal Circles**:
    - For each detected internal circle on the board, the algorithm computes its radius, essential for scoring.
8. **Distance Comparison**:
    - The distances from the center of each dart cluster to the center of the dartboard are compared to assess the scoring position of each dart.
9. **Scoring Points**:
    - Based on these distance calculations, the algorithm determines the points scored by each dart.
10. **Classifying Darts**:
    - Each dart is classified into one of two classes based on its color: red or green.
11. **Calculating Total Points and Declaring the Winner**:
    - Finally, the algorithm calculates the total points for each player and determines the winner based on the scores.

Explained Solution

Detecting the Largest External Circle

The image undergoes several pre-processing steps to prepare it for contour detection:

- **Reading the Image**: The image is read from the provided source path.

- **Grayscale Conversion**: The color image is converted to a grayscale image to simplify the processing.
- **Blurring**: A Gaussian blur is applied to the grayscale image to reduce noise and smooth the image.
- **Edge Detection**: The Canny edge detection algorithm is used to highlight the edges within the image, making it easier to identify contours.

After pre-processing, the program detects all contours in the image. It then searches for the largest contour that approximates a circular shape, using the following criteria:

- **Perimeter Calculation**: The contour's perimeter is computed.
- **Polygon Approximation**: The contour is approximated to a polygon to simplify its shape.
- **Circularity Check**: The circularity of each contour is calculated. Only those contours with a high circularity (indicating they are close to circular) are considered. This helps in accurately identifying the dartboard's contour

Locating the Center of the Dartboard

To accurately locate the center of the dartboard, the algorithm follows these key steps:

- **Find the Center**: Calculate the moments of the largest contour to find the center coordinates (cX, cY). An adjustment is made to cY to account for image distortions.
- **Validate Circularity**: Ensure the contour is sufficiently circular by checking its circularity, which should fall between 0.7 and 1.3

Color Segmentation of Darts

To segment darts by color, the algorithm follows these steps:

- Define Colors: Specific RGB values for red, green, and brown dart tips are defined and converted to HSV color space for better color detection.
- Set Color Ranges: HSV ranges are established for each target color: red, green, brown, and a specific target color.
- Generate Masks: The `find_colors` function creates masks by isolating pixels within the defined HSV ranges for each color.
- Clean Masks: The `mask_denoiser` function refines masks for better accuracy by removing noise through morphological operations.

Clustering Dart Tips

Function identifies and sorts the radii of dartboard rings from given contours. It calculates the average radius for each contour relative to the dartboard's center (`center_x`, `center_y`) and measures the distance to the center. These distances and radii are stored, sorted, and separated into lists. The function then computes differences between consecutive radii to identify distinct rings, highlighting those with significant spacing.

Distance Comparison and Scoring Points

For each detected dart centroid, the function calculates the Euclidean distance from the dartboard's center, represented by coordinates `cx` and `cy`. This distance is then compared against the radii to determine the appropriate scoring zone. The dart is scored higher if it lands closer to the center and progressively lower for outer rings. Specifically, if the distance of a dart falls within the innermost circle, it is awarded 80 points. Darts landing in successive outer circles receive 60, 50, 40, 30, 20, and 10 points respectively, based on the specific range they fall into.

Classifying Darts, Calculating Total Points, and Declaring the Winner

For classification, it defines specific BGR color ranges for red and green. Each dart's centroid, representing the point of impact, is analyzed by examining the pixels within the surrounding window. The function counts the number of pixels that fall into the red and green color ranges. If the count of red pixels exceeds that of green pixels, the dart is classified as red, and the corresponding score is added to the red player's total. Similarly, if green pixels predominate, the dart is classified as green, and the score is assigned to the green player. If neither color is predominant, the score is added to the 'other' category, though this typically doesn’t affect the game outcome.

After processing all darts, the function calculates the total scores for both the red and green players and prints these scores for verification. It then compares the scores to declare the winner: the player with the higher score is announced as the winner, while a tie is declared if both scores are equal.

Test algorithm on input image:

As we see, algorithms calculate 190 points for the Red team and 120 points for the Green team. Obviously that Red team win this game 🎉🎉🎉.

In conclusion, I want to thank you for developing such an interesting test tasks and I hope that you will give me more opportunities to improve myself as a specialist. Thank you in advance.