Shape Detection Project Report

Overview

This project aims to detect and identify geometric shapes (like circles, triangles, rectangles, and squares) in images by comparing them with a reference image. The main components of the code include loading images, preprocessing them, extracting contours, matching shapes, and displaying the results.

Code Functionality

1. **Loading the Reference Image**:
    - The reference image is read using OpenCV's cv2.imread() function.
2. **Preprocessing the Reference Image**:
    - The reference image is converted to grayscale to reduce computational complexity.
    - A Gaussian blur is applied to the grayscale image to smooth out noise and improve edge detection.
    - Instead of using the Canny edge detection method, **adaptive thresholding** is employed. This technique is effective for detecting edges in images with varying lighting conditions.
3. **Extracting Contours**:
    - Contours are extracted from the thresholded image using cv2.findContours(), which helps to identify the shapes present in the reference image.
4. **Setting Minimum Contour Area**:
    - A minimum contour area threshold (MIN_CONTOUR_AREA) is set to filter out small contours that may not be significant. This threshold can be adjusted depending on the size of the shapes being detected.
5. **Shape Identification Function**:
    - The get_shape_name() function analyzes the contours to determine the shape type based on the number of vertices and the contour's circularity.
6. **Matching Shapes**:
    - In the match_shapes() function, target images from a specified directory are processed in the same way as the reference image. Adaptive thresholding is again applied.
    - Contours are matched with those from the reference image using cv2.matchShapes(), which compares the similarity of the contours based on a defined method.

- When a match is found, the detected shape is highlighted in green, and its name is annotated in red above the shape.

7. **Iterating Through Test Images**:
    - The code loops through all images in a specified directory, processing each image to detect shapes and display the results using cv2_imshow().

Limitations

Despite its capabilities, the project encounters several challenges:

1. **Detection Limitations**:
    - **Shadows and Reflections**: The algorithm may struggle to detect shapes that are partially obscured by shadows or reflections. The adaptive thresholding might not effectively differentiate the shape from its background when shadowed.
    - **Complex Backgrounds**: If the target images have complex backgrounds or colors similar to the shapes, the contours may not be accurately detected, leading to missed detections.
2. **Parameter Sensitivity**:
    - The contour area threshold (MIN_CONTOUR_AREA) is critical; if set too high, small shapes will be missed. Conversely, if set too low, noise can lead to false detections.
    - The match threshold in cv2.matchShapes() is also a sensitive parameter. Adjusting this threshold affects the balance between false positives and negatives.
3. **Shape Variability**:
    - Variations in shape (e.g., deformation, rotation) may lead to mismatches during detection. The current method relies on contour shape similarity, which may not accommodate all geometric variations.

Improvements

To enhance the detection capabilities, consider the following adjustments:

- **Utilize Image Preprocessing Techniques**: Experiment with additional preprocessing techniques, such as histogram equalization, to improve contrast before applying thresholding.
- **Implement Morphological Operations**: Applying morphological operations (like erosion and dilation) could help eliminate small noise and close gaps in contours.
- **Fine-Tuning Parameters**: Conduct experiments to find optimal values for contour area and shape match thresholds through cross-validation with a range of images.