Shivam Prakash Jha

Task: To detect basic shapes (Squares & Rectanges) in an image. Display the name of the shape on the on the shape itself.

A) Steps Involved in the Shape Detection Code

1. **Include Necessary Headers**
    - Include the OpenCV and C++ standard library headers required for the program.

```
cpp
Copy code
#include <opencv2/opencv.hpp>
#include <iostream>
#include <cmath>
```

2. **Define Helper Functions**
    - Implement helper functions to compute the cosine of the angle between two vectors, the Euclidean distance between two points, and the curvature of an edge.

```
cpp
Copy code
double angleCos(cv::Point pt1, cv::Point pt2, cv::Point pt0);
double distance(Point2f point1, Point2f point2);
double curvature(Point pt1, Point pt2, Point pt3);
```

3. **Main Function**
    - Load the input image and check if the image is loaded correctly.

```
cpp
Copy code
int main() {
    Mat image =
imread("/home/kpit/build/3rd_half/shapes/images/basicShapes.png");
    if (image.empty()) {
        cout << "Could not open or find the image!" << endl;
        return -1;
    }
    detectShapes(image);
    return 0;
}
```

4. **Convert Image to Grayscale**
    - Convert the input image to a grayscale image for easier contour detection.

```
cpp
Copy code
void detectShapes(const Mat& image) {
    Mat gray, blurred, edged;
    cvtColor(image, gray, COLOR_BGR2GRAY);
```

5. **Apply Gaussian Blur**

- Apply Gaussian blur to the grayscale image to reduce noise and improve edge detection.

```
cpp
Copy code
    GaussianBlur(gray, blurred, Size(5, 5), 0.5);
```

6. **Edge Detection**
    - Use the Canny edge detection algorithm to find edges in the blurred image.

```
cpp
Copy code
    Canny(blurred, edged, 50, 150);
```

7. **Find Contours**
    - Detect contours in the edge-detected image.

```
cpp
Copy code
    vector<vector<Point>> contours;
    vector<Vec4i> hierarchy;
    findContours(edged, contours, hierarchy, RETR_EXTERNAL,
CHAIN_APPROX_NONE);
```

8. **Approximate Contours and Identify Shapes**
    - Iterate through each contour, approximate the shape, and identify it based on the number of vertices and other properties.

```
cpp
Copy code
    for (size_t i = 0; i < contours.size(); i++) {
        vector<Point> contour = contours[i];
        vector<Point> approx;
        double peri = arcLength(contour, true);
        approxPolyDP(contour, approx, 0.01 * peri, true);

        // Identify shapes based on the number of vertices
        if (approx.size() == 3) {
            putText(image, "Triangle", approx[0], FONT_HERSHEY_SIMPLEX,
0.5, Scalar(0, 255, 0), 2);
        } else if (approx.size() == 4) {
            // Further checks for rectangles, squares, and trapeziums
            Rect rect = boundingRect(approx);
            double aspectRatio = (double)rect.width / rect.height;
            if (aspectRatio >= 0.95 && aspectRatio <= 1.05) {
                putText(image, "Square", approx[0], FONT_HERSHEY_SIMPLEX,
0.5, Scalar(0, 255, 0), 2);
            } else {
                bool isTrapezium = false;
                for (int j = 0; j < 4; j++) {
                    double cosAngle = angleCos(approx[j], approx[(j + 2) %
4], approx[(j + 1) % 4]);
                    if (abs(cosAngle) > 0.1) {
                        isTrapezium = true;
                        break;
                    }
                }
                if (isTrapezium) {
```

```
                    putText(image, "Trapezium", approx[0],
FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 255, 0), 2);
                } else {
                    putText(image, "Rectangle", approx[0],
FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 255, 0), 2);
                }
            }
        } else if (approx.size() == 5) {
            putText(image, "Pentagon", approx[0], FONT_HERSHEY_SIMPLEX,
0.5, Scalar(0, 255, 0), 2);
        } else if (approx.size() == 6) {
            putText(image, "Hexagon", approx[0], FONT_HERSHEY_SIMPLEX,
0.5, Scalar(0, 255, 0), 2);
        } else if (approx.size() == 8) {
            bool isCurved = false;
            for (int j = 0; j < 8; j++) {
                double curv = curvature(approx[j], approx[(j + 1) % 8],
approx[(j + 2) % 8]);
                if (curv > 0.01) {
                    isCurved = true;
                    break;
                }
            }
            if (isCurved) {
                putText(image, "Rectangle", approx[0],
FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 255, 0), 2);
            } else {
                putText(image, "Octagon", approx[0], FONT_HERSHEY_SIMPLEX,
0.5, Scalar(0, 255, 0), 2);
            }
        } else {
            // Check for circles and ellipses
            double area = contourArea(contour);
            double circularity = 4 * M_PI * area / (peri * peri);
            if (circularity > 0.85) {
                putText(image, "Circle", approx[0], FONT_HERSHEY_SIMPLEX,
0.5, Scalar(0, 255, 0), 2);
            } else {
                RotatedRect ellipseFit = fitEllipse(contour);
                double aspectRatio = (double)ellipseFit.size.width /
ellipseFit.size.height;
                if (aspectRatio >= 0.5 && aspectRatio <= 1.5) {
                    putText(image, "Ellipse", approx[0],
FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 255, 0), 2);
                } else {
                    putText(image, "Polygon", approx[0],
FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 255, 0), 2);
                }
            }
        }
    }
```

9. **Display Results**

    - Show the annotated image with the identified shapes and wait for a key press to close the window.

```
cpp
Copy code
    imshow("Shapes", image);
    waitKey(0);
}
```

This step-by-step process ensures that each part of the program is well-defined and organized for efficient shape detection and classification in an image.

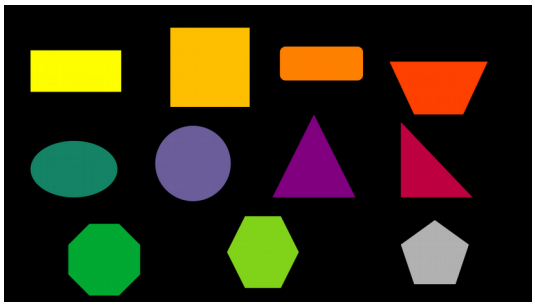
B. Outcome of Today's Work

Today, we implemented a shape detection application using the OpenCV library. The application can identify various geometric shapes such as triangles, squares, rectangles, trapeziums, pentagons, hexagons, octagons, circles, and ellipses in an image. The program processes the image, detects contours, approximates the shapes, and annotates the image with the name of the detected shape.

C. Difficulties Encountered

- **Shape Detection Accuracy**: Initially, there were issues with accurately detecting and classifying the shapes due to variations in contour approximation and shape curvature. This was addressed by fine-tuning the parameters for contour approximation and implementing additional checks for shape curvature and aspect ratios.
- **Image Not Loading**: There was an issue where the image file could not be found or loaded correctly. This was resolved by verifying the correct file path and ensuring the image was available in the specified directory.
- **False Positives and Misclassifications**: Some shapes were incorrectly classified due to similar contour structures. This was mitigated by adding more specific checks, such as angle cosine for trapeziums and curvature checks for distinguishing between octagons and other shapes.
- **Code Modularity**: Ensuring that the code was well-organized and modular was important for maintainability and debugging. This was achieved by separating the different functionalities into distinct functions and clearly defining their responsibilities.

Input Image



Output Image