Question 1, Gradient-Based Edge Detection & Boundary Tracing

What I solved -

- I created a gradient-based edge detector for grayscale images and smoothed the images using a Gaussian filter, with σ as a parameter. I also computed the horizontal and vertical derivatives to obtain gradient magnitude and orientation and traced object boundaries manually, starting from a pixel on the outer edge.

How it worked -

- I combined the derivatives in the x and y directions to calculate gradient magnitude.
- I ploted orientation using arrows.
- Boundary tracing involved walking along edge pixels from a known boundary point.

Errors faced-

- Noise caused broken or incomplete edges in some iterations and choosing the starting point for boundary tracing was hard.
- At times, traced boundaries missed small protrusions or corners.

Conclusion -

- The edge detector performed reasonably well on clear, high-contrast objects. The boundary tracing is tricky; most issues arised from noise or unclear edge pixels.
- Using more reliable tracing algorithms or automatic starting points could maybe improve the process.

Question 2, Harris Corner Detection

What I solved -

- I built the Harris corner detector and calculated the corner response for each pixel.
- I selected local maxima within a radius r as corner points.

How it worked -

- I use gradients to compute the structure tensor for each pixel. The corner response function highlighted points with high curvature.
- Non-maximum suppression identifies the most prominent corners.

Errors I faced -

- A small radius r sometimes led to duplicate corners near edges. Noise in images resulted in false positives.
- The computation slowed down for large images since I didn't use optimized functions.

Conclusion

- The detector found corners well in high-contrast regions. But performance dropped in noisy images or low-texture areas.
- Using adaptive thresholds or pre-smoothing could maybe give better results.

Question 3, Hough Transform (Lines & Circles)

What I implemented -

- I implemented the Hough transform to find straight lines and circles.
- myHoughLine() identifiesd the top n lines by voting in Hough space and myHoughCircleTrain() and myHoughCircleTest() detected circles of a specific radius.

How it worked -

- For lines, I transformed edge points into Hough space; votes indicated strong lines.
- For circles, I constructed reference data from known circle points and matched them in new images. I visualized results by plotting detected lines or circles on images.

Errors I faced -

- Voting sometimes missed faint or partially broken lines. Circle detection was sensitive to the radius parameter - small deviations led to false negatives.
- Overlapping shapes sometimes confused the algorithm.

Conclusion-

- Accuracy decreased with noise.