Cloverleaf Bridge

Computer Vision

Assignment 1

Varun Edachali (2022101029)

January 2025

1 Notes

Our goal is to detect and analyse the cloverleaf-like structures in a given image using classical image processing techniques.

Figure 1: the initial loaded image

1.1 Dataset and Preprocessing

1.1.1 Image Loading and Exploration

The provided cloverleaf image has the following properties:

property	value
shape	(1207, 1207, 3)
dtype	uint8
min intensity	0
max intensity	253
mean intensity	67.9

A histogram is calculated without any external dependencies (but `numpy`) using a simple frequency map and compared with `OpenCV`'s histogram function. On ensuring the two histograms are equivalent, we visualise each colour's intensity frequency (channel histogram).

The key difference between OpenCV's histogram calculation and ours is efficiency - opencv likely uses low-level vectorisation primitives that optimise the histogram calculation. The simple frequence mapping was calculated multiple orders faster by opencv.

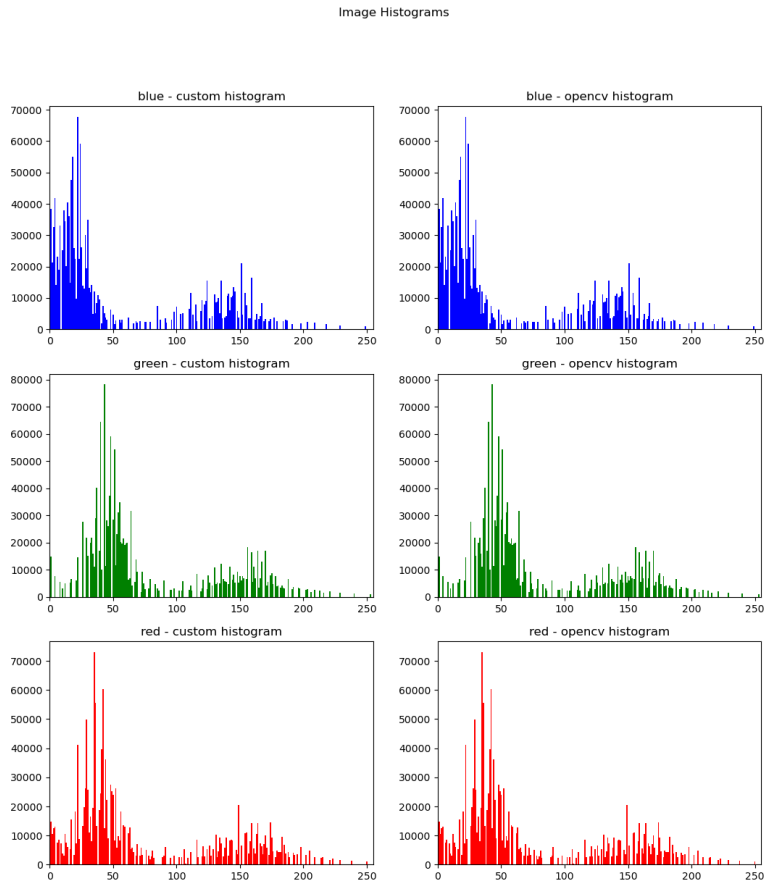

Figure 2: histograms of the red, green and blue channels of the image

1.1.2 Image Processing

In order to process the image to make it simple(r) to analyse the cloverleaves, we do the following, in order:

1. **colour the image gray** - since we are only concerned with the shape of some items in the image, and not their colour, we can simplify our image by reducing it to a single channel, i.e., to a grayscale image.

Figure 3: grayscale image

2. **blur the image** - in order to smooth out a few uneven pixel values, we perform a simple gaussian blur. This evens out some of the colour within the cloverleaf like structures themselves. Here, a small kernel was not adequately removing some small artifacts while a larger kernel was removing too much detail, so I stuck to $(5, 5)$.

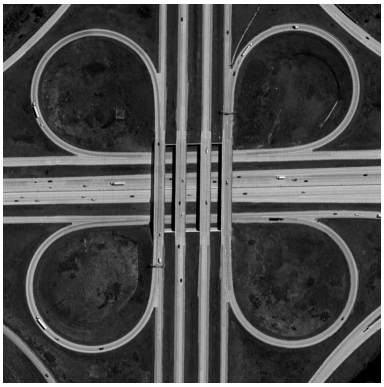

Figure 4: blurred image

3. **threshold the image** - again applying the same logic as before: we are only concerned with the shape of the road like structures, specifically

the ones that form cloverleaves. Thus, we can completely brighten these structures and dim the background fields by using thresholding. I used `Otsu`'s method to automatically select a threshold intensity value, and it worked as intended.

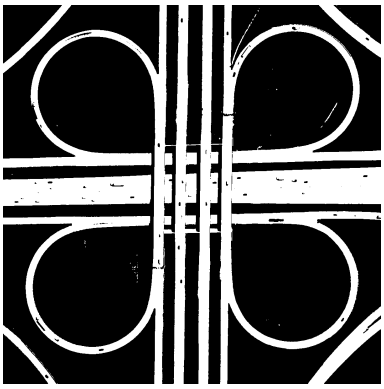

Figure 5: thresholded image

4. **closing the image** - in order to remove some of the peppering within the road structures, we perform a `close` operation as suggested in the OpenCV documentation.

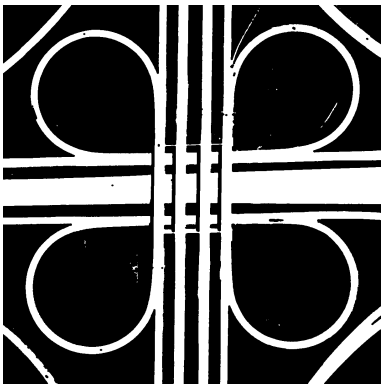

Figure 6: closed image

5. **opening the image** - in order to remove some of the spotted artifacts surrounding the road structures, we perform an `open` operation, as in the documentation.

   I used a larger kernel size of $(7, 7)$ because smaller ones did not adequately denoise the thresholded image.

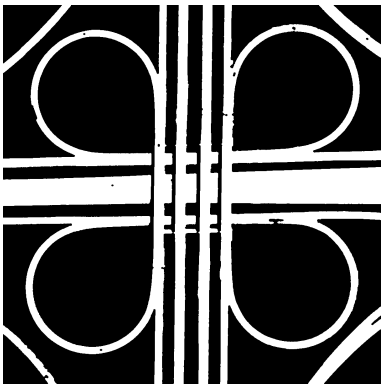

Figure 7: opened image

6. **dilating the image** - finally, if we observe the cloverleaves in the left half: they appear to be connected together because of a slight gap in the structure, likely due to some shadowing in the original image. In order to connect the white (road) structures and disconnect the cloverleaves in order to enable contour detection with ease, we dilate the image for a few iterations. Here, a single iteration was not sufficient to adequately "bridge the gap" between the roads, so I used 2 iterations. More than this led to too much dilation and the cloverleaves began losing their shape.

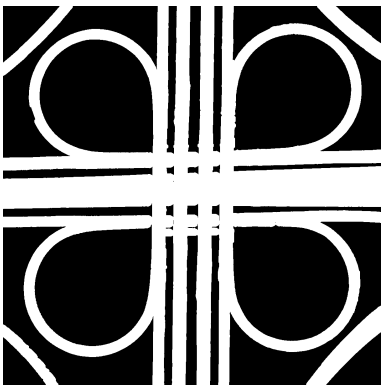

Figure 8: dilated image

1.2 Cloverleaf Detection and Measurement

Here, we leverage our pre-processed image to detect cloverleaves, estimate their radii and approximate the area they enclose.

1.2.1 Cloverleaf Detection and Visualisation

Here, I propose an algorithm for cloverleaf detection that should theoretically minimise both false positives and false negatives. It works as follows:

- *to minimise false negatives*: The idea here is to match the perimeter of a detected contour to its area. We do this by leveraging primitives to extract the area, get the estimated radius from it (assuming it is a cloverleaf), and check if the predicted perimeter is close enough to the actual perimeter of the radius. In order to ensure maximal matching of the perimeter and area, we perform these operations on the convex hull that surrounds the contour (since it is smooth and may be closer to a "true" cloverleaf than the noisy contour).
- *to minimise false positives*: there is a possibility that the convex hull of a structure resembles a cloverleaf, but not the structure itself, for example: if we have a complete cloverleaf apart from a "notch" in it. In order to prevent detecting these structures as cloverleaves, we ensure the area of the contour and its hull are within some reasonable threshold.

Note: I revealed this thought process of mine to Samyak Mishra (2022101121) before implementing it, but did not share any code.

Thus, we detect the contours in the pre-processed image and use the above algorithm to filter out the cloverleaves we seek.

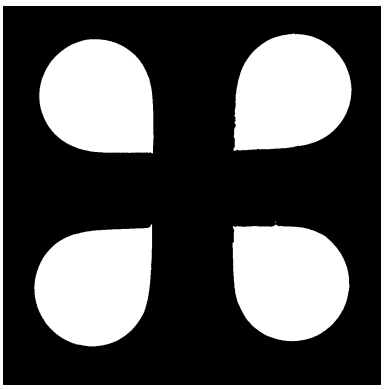

Figure 9: detected cloverleaves

There is some minor noise due to very small contours being detected as cloverleaves, in order to get rid of this, we perform an `open` and a `close`, as before:

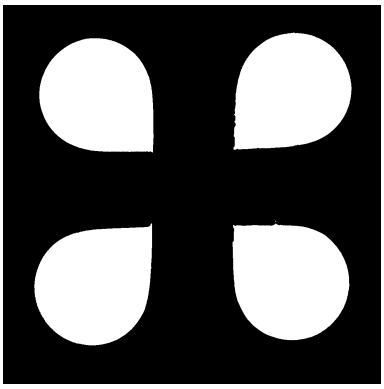

Figure 10: detected cloverleaves - closed

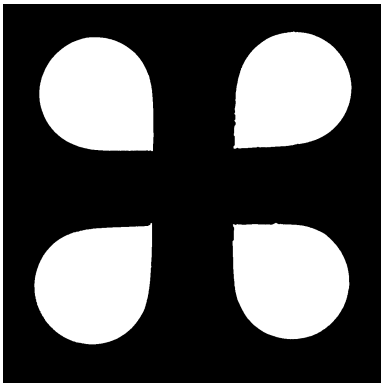

Figure 11: detected cloverleaves - opened

This yields our final detected cloverleaves.

Note: this method depends heavily on the assumption that the cloverleaves do consist of a $\frac{3}{4}^{th}$ circle and a square, such that they obey the following formulae for some radius r:

$$area = \frac{3}{4} \cdot \pi \cdot r^2 + r^2$$

$$perimeter = 2 \cdot r + \frac{3}{4} \cdot 2 \cdot \pi \cdot r$$

In addition, the method heavily depends on good pre-processing. If we do not pre-process the image, and directly apply the above method on the grayed version of the image, then the detected cloverleaves are as below: Also, here are the results of detecting hough circles on the given (raw) image, with the same parameters as the processed image (described below):

The primary difficulties in detecting the cloverleaves lay in isolating them - initially the detected contours were attaching them and making it infeasible for

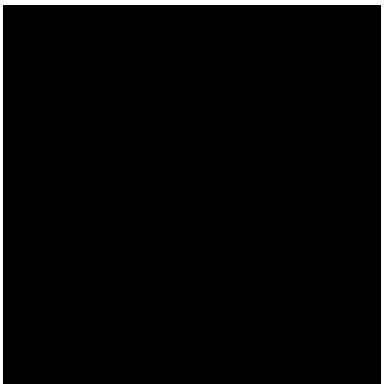

Figure 12: detected cloverleaves - on grayed

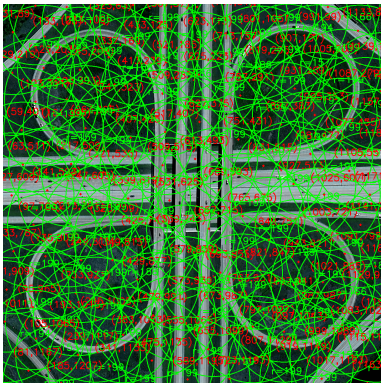

Figure 13: hough circles on raw image

accurate detection. After some noise removal and dilation, the results improved considerably.

The pipeline as a whole will find difficulty in low-res images that cannot be easily thresholded using straight-forward methods like `Otsu`'s. If the cloverleaves cannot be well isolated such that the contour detection can catch them, they will not be easily identified. As an example, attached is the result of thresholding on a low-res image I found online:

Figure 14: low res cloverleaf bridge

Figure 15: low res cloverleaf bridge - thresholded

1.2.2 Measuring the Radii of the detected cloverleaves

Since our final detected cloverleaves are highly de-noised and only contain the cloverleaves themselves, we can leverage `Hough Transforms` in order to estimate circles that fit within the cloverleaves. The centers and radii of these detected circles form our estimated radii.

We overlay these circles onto the original, raw image, in order to visualise our estimates as shown.

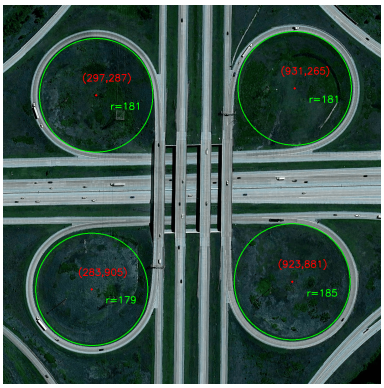

Figure 16: estimated radii of the detected circles

The accuracy of this estimation method depends heavily on the accuracy of the cloverleaf detection method. If our previous method significantly affects the cloverleaf structure, then the corresponding radius estimation may be affected.

The parameters passed to the `HoughCircle` were largely arbitrary, but my understanding of them is as below:

- $dp = 2$ is the inverse ratio of the accumulator resolution to the image resolution. For an image of roughly $(1200, 1200)$ (ours), this creates an accumulator with a $600 \cdot 600$ grid. This lower resolution helps in finding less precise but more significant circles. Since our circles are (expected to be) large and significant in the image, this value is feasible.
- $param1 = 50$ is the higher threshold of the two passed to the canny edge detector, which controls the sensitivity of edge detection. A value of 50 is moderate, it will detect significant edges while ignoring minor noise: which is why it works for our clear, high contrast image with large circles.
- $param2 = 30$ is the accumulator threshold for circle detection, representing the minimum number of intersections to detect a circle. A value like 30 is lower than usual, which is useful for detecting a few, but distinct circles. Since there are large, clear circles this is useful, but if we had a noisy image we would likely have a lot of false positives.
- $minDist, minRadius$ and $maxRadius$ were selected based on the image at hand - with large, prominent circles at some separation.

I learnt about Hough Circles through this video, and by asking for an explanation from perplexity, which explained that first, edge pixels are detected using techniques like Canny edge detection. Then, for each edge pixel, potential circle centers are hypothesized by testing different radii. These potential centers accumulate votes in an accumulator space. Locations with high vote counts represent likely circle centers, with the highest peaks indicating the most probable circular shapes in the original image.

1.2.3 Approximating the Area enclosed by each cloverleaf

We use three distinct methods to estimate the area enclosed by each cloverleaf.

- *Using the estimated radii*: we use the estimated radii to directly estimate the expected area of the cloverleaf based on the formula from above. This assumes a true cloverleaf.
- *Using the contour's area*: instead, we can directly get the area enclosed by the contour that has been detected as a cloverleaf. This is a more true representation of the structure's area within the image itself.
- *Using the enclosing hull's area*: if we want to assume that our cloverleaf is closer to a "true" cloverleaf, free of deformities along the perimeter, we can extract the area of the surrounding hull.

I believe the second method is the most accurate to extract the exact area truly covered by each contour. The first method is feasible if we assume a "true" cloverleaf, while the last is good if we want a good approximation of how a smooth cloverleaf-like structure would look closest to our source.