CSU44053 Computer Vision | Assignment 1
Brendan McCann | 20332615

1 Theory of the Processing

1.1 Smoothing

To help cope with noise, all my images went through an initial stage of smoothing. In the context of this problem, it was crucial that the edges of the pedestrian crossings remained well-defined and as sharp as possible. The reason for this is because I wanted my edge detection to work as good as possible. As a result, I settled with **median smoothing**, primarily because it does not blur edges too much.

1.2 Edge Detection

At first, I experimented with region processing using Mean Shift Segmentation (MSS). However, I found that it took a about two seconds to process the image using MSS. In the context of the problem, I opted for a more efficient approach as it is vital that images are produced rapidly in a car imaging system. This is how I settled with edge processing.

I chose to go with **Canny Edge Detection** as opposed to Boundary Chain Codes, because with Canny I could have more customisation over the results by changing the parameters, whereas with Boundary Chain Codes it was more limiting.

Figure 1: Median Smoothing & Canny Edge Detection. I experimented with multiple iterations of median smoothing, however I found the best results with just one iteration. Additionally, I opted for a filter size of 3x3 as I wanted the edges to be as sharp as possible. With larger filter sizes, I was worried it would corrupt the image too much and deviate too far from the original. For the edge detection, the low and high thresholds were chosen mostly based on trial-and-error. I ensured that my low threshold value was conservative, as this is indicative of the number of contours detected by Canny, and I wanted to be sure that my edge detection would not miss out on any potential pedestrian crossings.

1.3 Binary Thresholding

The output of the previous step in the process was an RGB multispectral image, and I wanted a binary image to simplify things further. I used **Otsu Thresholding** to convert to binary, followed by a closing operation.

1.4 Connected Components Analysis and Shape Analysis

My next goal was to find contours in the binary image. I used CCA for its simplicity, and also because it showed great results on my initial testing.

After finding all the contours in the image, they would then go through the following filtering process:

All parameters were chosen through trial-and-error testings.

1. Filter by Contour perimeter and area
    - Contour Perimeter >= 90
    - 50 <= Contour Area <= 3000
2. Convex Hulls were then drawn around these contours. The Convex Hulls were filtered by area.
    - 500 <= Convex Hull Area <= 3500
3. Convex Hulls were then filtered by rectangularity.
    - Rectangularity >= 0.5

After going through this filtering process, we now have a list of filtered convex hulls that potentially identify as pedestrian crossings.

Figure 2: Left-to-Right: Otsu Thresholding, Closing, CCA & Shape Analysis. The closing operation was done to fill any potential gaps in the edges of pedestrian crossings. This would yield better results for CCA. From some trial-and-error testing, I found a closing kernel size of 2 to be optimal for my process.

1.5 Colour

One distinctive trait of pedestrian crossings are it's white-ish colour. For each filtered convex hull, the mean colour of the convex hull is calculated. This RGB value is compared to the white RGB value by calculating the Euclidean distance between the two 3D vectors. From trial-and-error testing, I found 150.0 to be a forgiving value, but with good results.

1.6 Filtering out isolated & overlapping Convex Hulls

Another distinctive trait of pedestrian crossings is that they are positioned relatively close together. In this step of the process I wanted to filter out convex hulls that were isolated and had no neighbours - likely these are not pedestrian crossings because otherwise they would have neighbours.

This was achieved by finding the centre points of each convex hull and calculating its distance to the other convex hulls. If it had no convex hull within a distance of 175 (found through testing), it was filtered out.

Some convex hulls were overlapping with each other, so these were filtered out by introducing a minimum distance value of 10.

Figure 3: Left-to-right: Convex Hulls, White Regions, Filtering out isolated convex hulls. We can see the centre image has filtered out the van region in the top-right of the image because my process has deemed it is not white enough in colour to be determined as a pedestrian crossing. In the right image we can observe the leaves in the bottom right corner have been filtered out as they have no nearby neighbours. If you look closely at the pedestrian crossings, we can observe that overlapping edges have also been filtered out.

1.7 Finding the longest linear sequence

A key trait of pedestrian crossings is that they occur in a straight line. The aim of this step was to determine the longest linear sequence among the potential pedestrian crossings. This was achieved with a brute-force approach - in reality, there will only be a handful of potential pedestrian crossings so any efficiency improvements would be negligible.

A line is drawn from the centre of one convex hull to a second convex hull. Then, another line is drawn from the second convex hull to a third convex hull. If the angle between these two lines is less than 8°, it is deemed in a straight line and the algorithm continues to find other convex hulls that also lie within this threshold. This is repeated for every possible combination of convex hulls, and then the longest linear sequence is obtained.

Additional Heuristics

- If there is a tie for the longest sequence, my algorithm will choose the linear sequence that is 'straighter'.
- A linear sequence must be within a 30°angle to a horizontal line. This was done to prevent road markings from being classified as pedestrian crossings as they often run vertically on the screen.

- A linear sequence must be of length >= 3 to be considered a pedestrian crossing.
- My algorithm will only return one potential pedestrian crossing. If there are multiple pedestrian crossings in the image, it will choose the longer one.

1.8 Drawing the Bounding Box

If a pedestrian crossing was identified, the bounding box must be drawn around it to compare with the ground truth.

Various straight lines were drawn through the centres of the pedestrian crossings and the mean slope was calculated. The top edge of the bounding box would be drawn through the maximum y-coordinate with this mean slope, and the bottom edge of the bounding box would be drawn through the minimum y-coordinate with this mean slope.

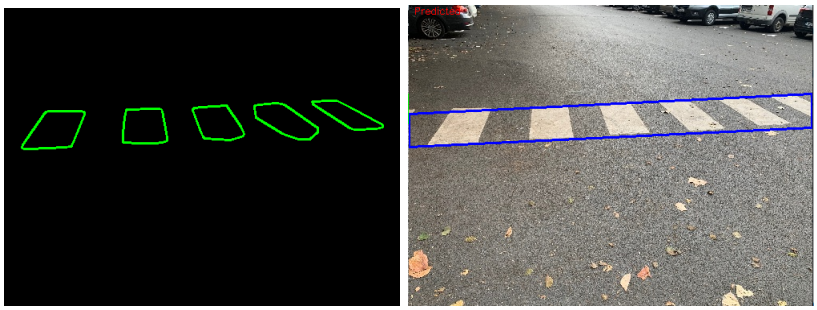

Figure 4: Left: Longest Linear Sequence. Right: Drawing the bounding box. My algorithm has determined all 5 convex hulls to be in a relatively straight line to each other.

2 Results and Analysis

Performance Metrics

Seeing as this is an object detection problem, it made sense to use Intersection over Union (IoU) as my performance metric.

$$IoU = \frac{Predicted \cap Ground\ Truth}{Predicted \cup Ground\ Truth} = \frac{Intersection\ Area}{Ground\ Truth\ Area + Predicted\ Area - Intersection\ Area}$$

The *Intersection Area* was calculated using the `intersectConvexConvex` method. The *Ground Truth Area* and *Predicted Area* were calculated using the `contourArea` method.

Following from the advice in the lectures, an IoU value over 50% is considered a good match.

Additionally, the F1 score will be used in measuring my algorithm's performance based on both training and test data. For the sake of simplicity, an observed positive is when the IoU is over 50%. I chose to go with this metric as it integrates elements of both Precision and Recall.

$$F1\ Score = \frac{2 * TP}{2 * TP + FP + FN}$$

2.1 Successes

I will not be including all cases of successes (as many of them are trivial), only examples of significance. The green lines correspond to the Ground Truth, and blue lines correspond to my prediction.

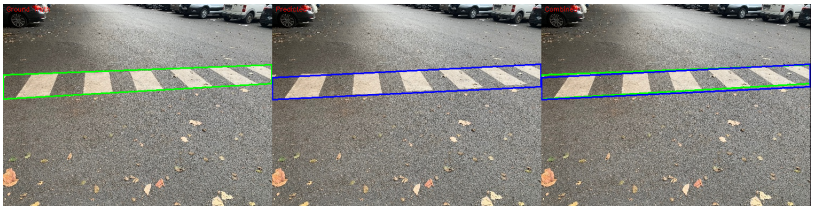

Figure 5: Left: Ground Truth, Middle: Predicted, Right: Combined. The algorithm performs very well with IoU scores of 87%, 73%, 90%, 88%, 88%, 86%, 78%, 89% respectively.

2.2 Failures

2.3 Robustness