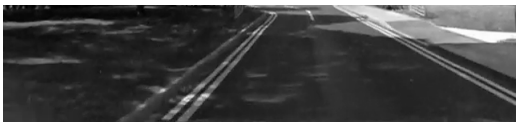
Software, Systems and Applications III – Computer Vision

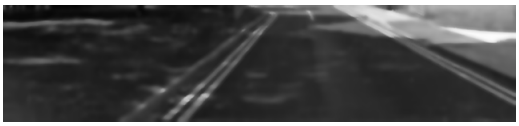
Tom Robson - hzwr87

Preprocessing

Following an examination of the dataset, it was determined that most of the foreground edges required could be found in the lower 30% of the image. Therefore, the top 70% of the image was cropped out, as this removed much of the noise, and therefore many of the potential false positives. Next was the conversion to grayscale. Conversion to other colour spaces were attempted but these gave an inferior output when used for edge detection. Another approach attempted for this was to identify the dominant colour in the road lines, and filter out all pixels that did not have this dominant colour. However, this did not prove to be very effective. The next step was to apply blurring to the image using a Bilateral Filter, as this gave the best results experimentally. The combination of these simple preprocessing techniques gave the best results to the edge detector of all the techniques attempted, many of which were more complex.



(a) Grayscale

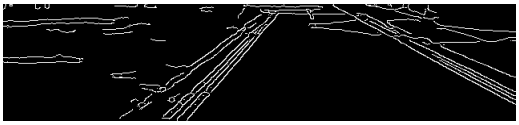


(b) Bilateral Filter on Grayscale

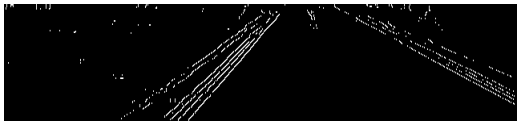
Figure 1: Preprocessing

Edge detection

Following the preprocessing of the images, the next step was to pass them to an edge detector. In this case, the OpenCV Canny function was used. In order to optimise this, an auto-canny method was implemented to adapt the thresholds to suit each image by finding the median of the grayscale pixel values of the filtered image, and multiplying this by 0.67 for the lower threshold or 1.33 for the higher threshold. On examination of the output of this Canny function, it was clear that further processing was necessary. Morphological opening was implemented to remove much of the noise while leaving the road lines relatively unharmed.



(a) Simple Canny



(b) After morphological opening

Figure 2: Edge Detection

RANSAC

First, two of the non-zero pixels in the canny image are randomly selected, and a line is drawn between them. The gradient of this line was then calculated, and used to extend the line over the whole of the bottom 30% of the image. Due to the fact that the vast majority of the foreground lines that are of interest to us are close to vertical, any line with a gradient close to horizontal was eliminated. This greatly reduced the number of false positives returned by the algorithm. This line is then drawn onto a blank image, and compared to the canny image using a bitwise AND operation. If the number of common pixels is above the threshold, then the line is accepted. Then, for each of the accepted lines, we compare them to existing lines drawn on the image and check if their gradient is too similar, if the lines cross over each other or if their x and y coordinates are too close together. This gives a more accurate count of the lines found per image, rather than having RANSAC find many lines associated with the same road line in the actual image. This gives the results shown in the output files. After all of the trials have been done, if no acceptable line has been found the algorithm is run again with a lower threshold to try and find a line. This is run until a line is found, or until the threshold gets too low, in which case it is accepted that no line is found. This approach slows down runtime and generates some false positives, but also finds more valid lines.

(a) All lines found

(b) Lines filtered down

Figure 3: Comparison of RANSAC outputs

Evaluation

Across the majority of the data set, this implementation performed well. On the images with clear road lines, it found at least 1 per image, as seen in Figure 4 and Table 1. Most of the lines that it does not find are lines that it would not be expected to find, such as short snippets of lines, lines covered by shadow or lines that are poorly marked.

(a)

(b)

(c)

(d)

Figure 4: Successes

However, some problems were encountered, and some of these are shown in figure 5. Images (a) and (b) are thrown off by noise, creating false positives, but were also unable to find the roadlines in these images. In these cases,

this was due to poorly marked roads. The line in image (c) has been influenced by the car, and also illustrates a drawback of drawing the lines across the whole bottom 30% of the image, as it extends further than it should, whereas (d) has two correct lines and one incorrect line caused by noise.

(a) (b) (c) (d)

Figure 5: Problems

This video https://www.youtube.com/watch?v=n_9_PGpW0rI shows the algorithm running through the entire dataset.On most of the images it performs well, but there are occasional false positives, mostly due to the lowering of the threshold if no lines are found. These are found in the more challenging images.

Image Number (randomly selected)	Lines Correctly Found	Lines Missed	Incorrect Lines
402	2	1	0
326	2	2	0
61	2	1	0
3	1	2	0
239	1	2	0
170	2	0	0
184	1	2	0
498	2	1	0
183	2	1	0
185	2	0	0
67	1	2	0
451	2	1	0
256	2	0	0
434	0	0	2
322	2	1	0
418	0	0	1
232	2	1	0
359	2	1	0
215	1	1	1
78	2	1	0
514	2	1	0
408	2	1	1
405	2	1	0
168	1	2	0
26	1	2	0

Table 1: Quantitative analysis of performance