FINDING CAR LANE LINES USING COMPUTER VISION

I wrote an image processing algorithm using computer vision to detect lane lines on a road. This algorithm could potentially be used by self-driving cars for example to help detect the lanes where the car should be steered.

The inputs of the program is a single image, in this case an image of a highway. The program will then output that same image with the lane lines of the road colored in red. This program can then be fed into a Video Buffer that successively takes a video frame by frame, calls my program, to produce a new video with detected lane lines colored in. My algorithm is written in Python and uses libraries from OpenCV (opencv.org).

I first considered simple detection techniques such as Color Selection. For example, one could detect the color white for example. This technique would be problematic because we could still have some other objects detected around the periphery that aren’t the lane lines that could also be the color white. Moreover, lane lines are not always the same color, and even lines of the same color under different lighting conditions (day, night, etc) may fail to be detected by our simple color selection. What we need is to take our algorithm to the next level to detect lines of any color using sophisticated computer vision methods.

Canny Edge Detection’s goal is to identify the boundaries of an object in an image. To do that, we must:

1. First convert to gray scale.

2. Then compute the gradient (convert the image such that the brightness of each pixel

corresponds to the strength of the gradient at that point). We are going to find the edges by tracing out the pixels that follow the strongest gradients. Rapid change in brightness is where we will find the edges.

By detecting the edges, we can more easily detect objects by their shape. I used the OpenCV Canny function:

*edges = cv2.Canny(gray, low\_threshold, high\_threshold)*

You can think of the strength of an edge as being defined by how different the values are in

adjacent pixels in the image i.e. the strength of the gradient. Our image is just a mathematical function of x and y so we can perform mathematical operations on it like any other function i.e. take its derivative, which is just the measure of change of this function. A small derivative means a small change. A big derivative means a big change. In computing it, we're measuring how fast pixel values are changing at each point in an image, and in which direction they are changing the most rapidly.

So I've now taken a grayscaled image, and using edge detection, turned it into an image full

of dots, but only the dots that represent edges in the original image. Now let's connect the

dots. I could connect the dots to look for any kind of shape in my image, but in this case I'm

looking for lines. To find lines I need to first adopt a model of a line, and then fit that

model to the assortment of dots in my edge detected image. Keeping in mind that my image is just a function of x and y, I can use the old familiar equation of a line y = mx + b. In this case,

my model includes two parameters, m and b.

In image space, a line is plotted as x vs. y. However, we want to convert the lines in parameters space, to “House space” using the Hough Transform to find lines from canny edges. In Hough space, I can represent my "x vs. y" line as a point in "m vs. b" instead. So, the characterization of a line in image space will be a single point at the position (m, b) in Hough space. So our strategy to find lines in image space will be to look for intersecting lines in Hough space. We do this by dividing up our Hough space into a grid, and define intersecting lines as all lines passing through a given grid cell. To do this, I'll first run the Canny edge detection algorithm to find all points associated with edges in my image. I can then consider every point in this edge-detected image as a line in Hough space. And where many lines in Hough space intersect, I declare I have found a collection of points that describe a line in image space. I will be using the OpenCV function HoughLinesP like this:

*lines = cv2.HoughLinesP(edges, rho, theta, threshold, np.array([]), min\_line\_length,*

*max\_line\_gap)*

Lastly, my algorithm performs Region Masking to focus on regions of the image that interest us. We can assume that the camera that took the image is mounted at a fixed position on the front of the car. Therefore, the lane lines will always appear in the same general region of the image. Adding a criterion to only consider pixels in the region where we expect to find the lane lines will only increase the accuracy of the algorithm.

The detect\_lane\_lines.py contains the source, and white.mp4 and yellow.mp4 are sample output videos using the lane detection algorithm.