The main goal of this homework is to apply a lane detection pipeline using image processing techniques on three different test images.

Here's an explanation of the main functions within the code:

Gaussian Blur (gaussian_blur):

This function blurs an image to reduce noise and detail using a Gaussian filter. It creates a Gaussian kernel, which is a square array of weights that corresponds to the Gaussian function, based on the specified kernel_size and sigma. The function then convolves this kernel with the input grayscale image to produce the blurred image. Blurring is an essential preprocessing step for edge detection as it reduces the impact of noise that can lead to false edges.

Canny Edge Detection (canny_edge_detection):

This function detects edges within the blurred image using the Canny edge detection algorithm. It first calculates the gradient magnitude and orientation using the Sobel operator. Non-maximum suppression is implemented to thin out the edges, ensuring that only the strongest edges are preserved and that they have a minimal width. It then applies a double threshold to determine potential edges and uses edge tracking by hysteresis to link edges together, distinguishing strong edges from weak ones.

Hough Transform (hough_transform):

The function performs the Hough Transform, a feature extraction technique for detecting straight lines. It maps points in the image space to curves in the Hough parameter space and looks for intersections in this space, which correspond to lines in the image space. An accumulator array is used to count votes for each line, with rho representing the line distance from the origin and theta representing the line orientation. The function then applies a threshold to identify significant lines within the accumulator space.

Draw Hough Lines (draw_hough_lines_on_image):

After identifying significant lines via the Hough Transform, this function draws these lines onto the original image. It converts the polar coordinates (rho, theta) from the Hough space back into Cartesian coordinates to draw the lines using OpenCV's line function. The resulting image visually represents the detected lines superimposed on the original road scene.

Processing and Saving Images (process_and_save_images):

This function ties all the previous steps together for a given image path. It reads the image, converts it to grayscale, applies the Gaussian blur, performs Canny edge detection, and then applies the Hough Transform. It saves the images resulting from each step into an output directory with filenames indicating the step and the original image name.

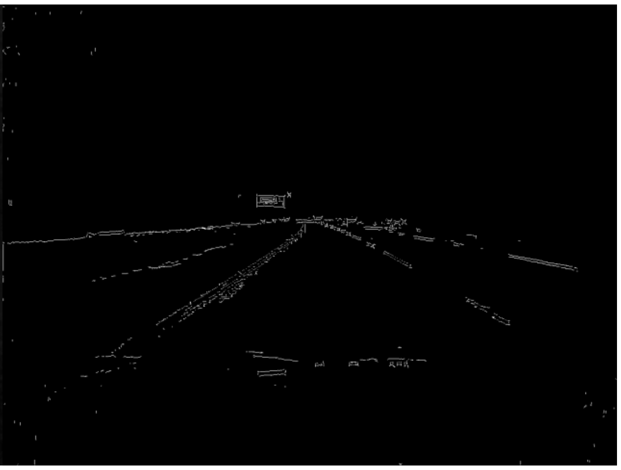
Batch Processing (process_images_in_directory):

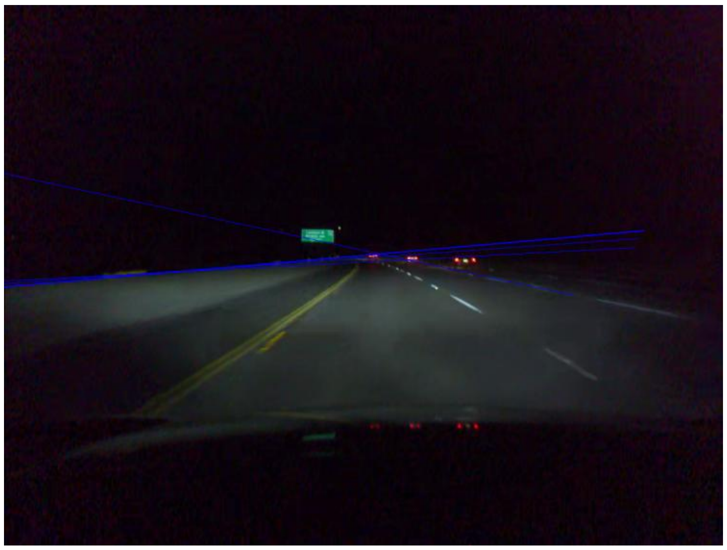
This function is designed to automate the processing of multiple images in a directory. It iterates over each image file in the specified directory, applies the entire lane detection pipeline, and saves the results.

Input images:

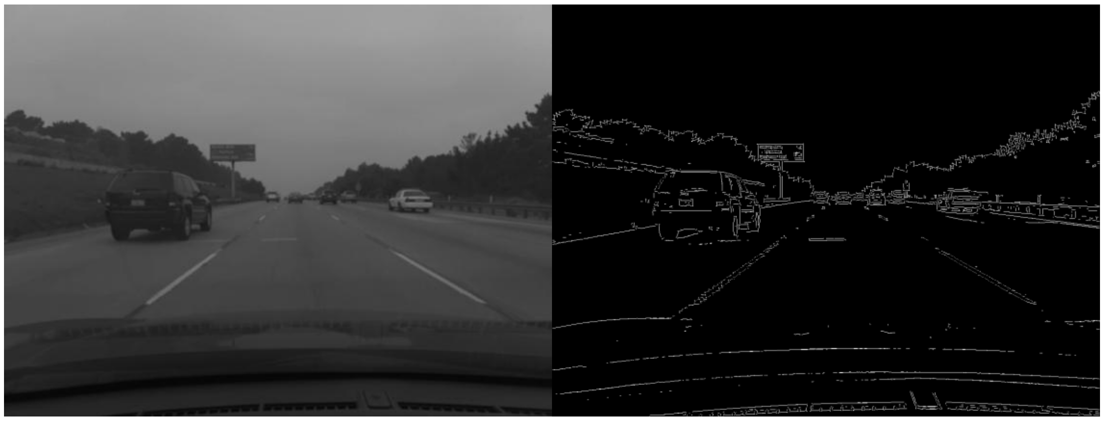
Output images (results):

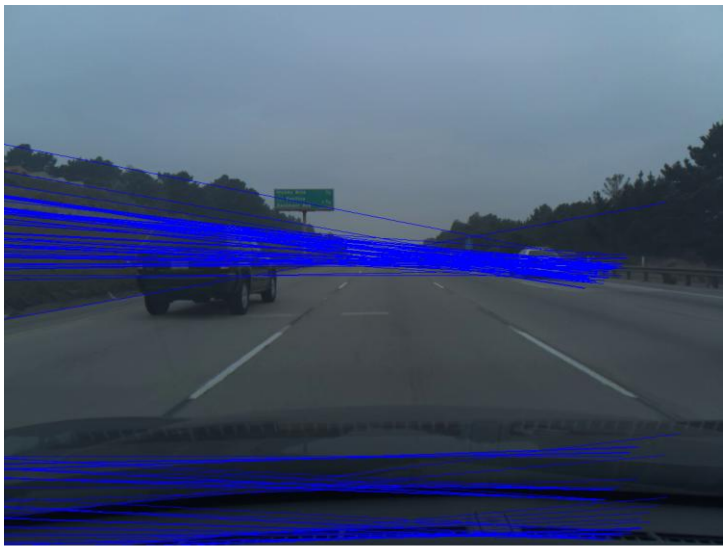
img1:



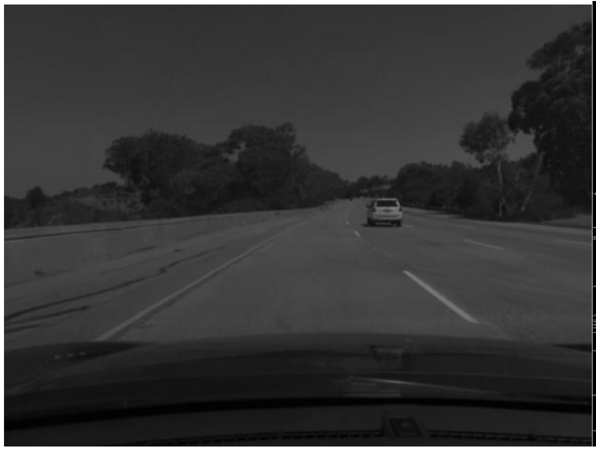


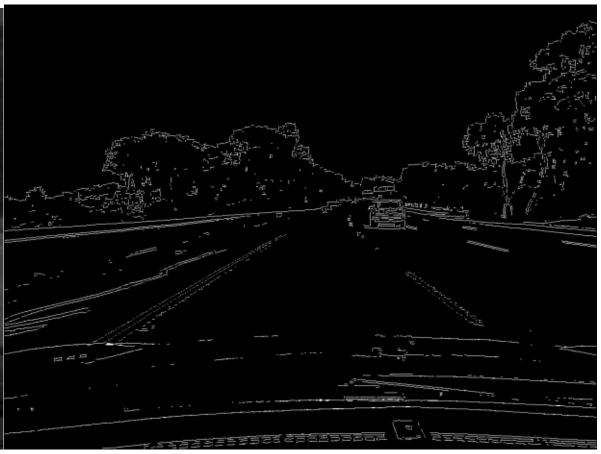
img2:

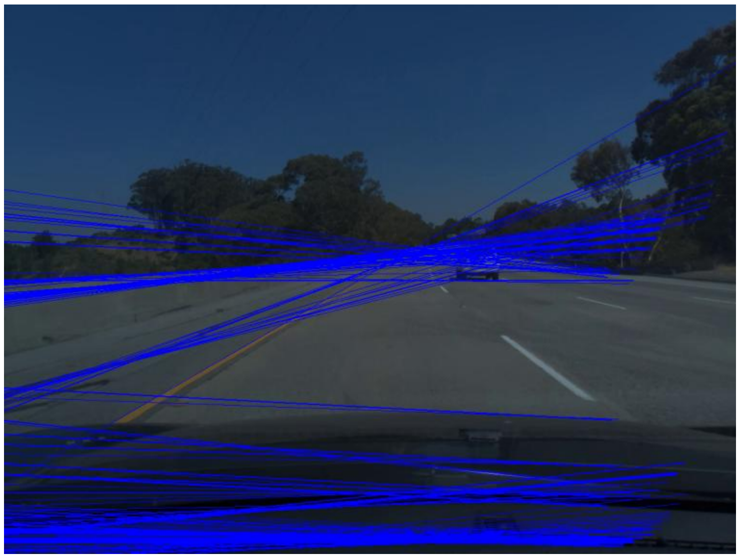




img3:







Explanation of the results:

The blurring step appears to have been correctly implemented, given the smooth appearance of the _q1 images.

For the Canny edge detection, the _q2 images show clear edge features, which indicates that the edges were detected successfully.

The Hough Transform step, resulting in the _q3 images, seems to have been less successful. The ideal output should show the detected lanes with a few solid lines rather than the multitude of lines that appear in the images. This suggests that the thresholds for the Hough Transform and possibly for the Canny edge detection could be too low, leading to many false positives. Fine-tuning these parameters should help to isolate the actual lane lines more effectively.