# Computer Visual Homework 3

ID:113598045 Name:林翡

- Explain your code:

1. **Convert to grayscale image and Gaussian Filter (convertGray、padding、Gaussian_kernel、Gaussian_filter Functions)**

   These functions are the same as those in homework 1 and homework 2. The purpose of the convertGray function is to calculate the gray value according to the formula (0.3 * R) + (0.59 * G) + (0.11 * B) and convert the RGB image into gray level image.In the padding function, the goal is to add padding around a given image.In the Gaussian_kernel function, the goal is to generate a 2D Gaussian kernel.In the Gaussian_filter function, uses a Gaussian filter to blur or smooth an image using a convolution with a Gaussian kernel.

2. **Canny Edge Detection (Canny_edge_detection Function):**

   In the Canny_edge_detection function, the goal is to implement Canny edge detection, which is divided into three steps: gradient calculation, non-maximum suppression, and double threshold and edge tracking by hysteresis.

   Step 1, use the Sobel operator to calculate the gradients in the horizontal direction (Gx) and the vertical direction (Gy) respectively, and specify the use of a 3x3 Sobel kernel for the convolution operation [Lines 82-83]. Then calculate the gradient magnitude [Line 85], and use the arctangent function to calculate the gradient angle. Convert units to degrees (range 0-180 degrees) [Line 86] and map directions to the interval [0, 180) [Line 87].

   The second step is to eliminate non-edge pixels and retain the maximum value points of the edges. First, create a new array with the same shape as the magnitude array, but with all values initialized to 0, to store the results of non-maximum suppression [Line 90]. Use a nested for loop to iterate each pixel (excluding the boundaries) [Lines 92-93]. According to the gradient angle of each pixel, determine its direction, 0° direction (Horizontally, you need to compare the pixel values on the left and right sides), 45° direction (bottom-left and top-right), 90° direction (bottom and top), and 135° direction (top-left and bottom-right) [Lines 95-104]. Then the pixel value is compared. If the pixel value is greater than the pixel value on both sides (maximum value), the pixel value is retained in nms. Otherwise, the pixel value is suppressed to 0 [Lines 106-109].

The third step is to classify edges into "strong edges" and "weak edges" and track weak edges connected to strong edges. First define the strong edge and weak edge pixels as 255 and 50 respectively, and then create a new array with the same shape as the nms array [Lines 111-113].
Points with pixel values greater than high_threshold are marked as "strong edges" (255), and points with pixel values between low_threshold and high_threshold are marked as "weak edges" (50) [Lines 115-119].Use a nested for loop to iterate each weak edge pixel of the entire image (excluding borders), if it is a weak_pixel, check whether its 3*3 neighborhood contains strong_pixel. If it does, update it to strong pixel. Otherwise, set it to 0 (weak edges are discarded) [Lines 122-129].

3. **Hough Transform (hough_transform Function):**

First, get the height and width of the Canny edge detection image [Line 133]. Next, determine whether the image needs to be masked to the upper half (set the pixels to 0). Because the lane lines of some images are concentrated in the lower half, the upper half is ignored and not detected [Lines 134-136]. Calculate the diagonal length of the image diag_len, which is the maximum possible value of ρ [Line 137]. Create a discrete range of values for rhos value and thetas, where thetas is converted to radians [Lines 138-139]. Create a counter matrix and initialize it to 0 [Line 142]. Find the coordinates of all non-zero pixels in the image (i.e. edge points) [Line 144], iterate each edge point, and calculate the rho values corresponding to different theta according to the formula: $\rho = x \cdot \cos(\theta) + y \cdot \sin(\theta)$ [Lines 145-147]. Correspond the calculated rho value to the discrete rho value (find the closest index) [Line 148], and add 1 to the value in the cumulative counter matrix of the rho value and theta value corresponding to the point [Line 149].

4. **draw_lines Function:**

According to the result of Hough transform, draw the detected straight line on the original image.

First copy the original image [Line 153], use a for loop and “np.where(counter > threshold)” to iterate to find locations where the cumulative count exceeds the threshold, which correspond to valid rho and theta values [Line 154]. Convert polar coordinates rho and theta into Cartesian coordinates of two points (x1,y1) and (x2,y2) [Lines 155-163]. Use the cv2.line function to draw this straight line on the image. The color of the line is red [Line 165].

**5. Main Function:**

First read three original images and convert them into grayscale images [Lines 170-176]. Perform Gaussian filter on the three grayscale images, the kernel size is all 5*5, the stride is all 1, and save them [Lines 178-184]. Canny edge detection will be implemented on the image after Gaussian filtering. The low_threshold and high_threshold of gaussian_img1 and gaussian_img3 are both 30,180, while the low_threshold and high_threshold of gaussian_img2 are 30,80 [Lines 186-188]. Because the lane lines of img1 are distributed in the lower half of the image, I enhanced the lower half of canny_img1 and set low_threshold and high_threshold to 20,100 to make the lower half of the image stand out more [Lines 191-196]. The lower right corner region of canny_img2 image is also specially processed. The lower right corner region of canny_img2 is extracted, enhanced, and then the enhanced result is covered to the same area of the original image [Lines 199-205]. For the processing of canny_img3, directly set the upper 1/4 pixels of canny_img3 to 0 to shield the edges of this area [Lines 208-210]. Then save three images after Canny edge detection [Lines 212-214].

Next, the image after Canny edge detection is used to detect straight lines by Hough transform. There are some differences in the parameter settings (theta_res rho_res, theta_min, theta_max, mask_top) of calling the hough_transform function [Lines 216-218]. According to the Hough transform result, draw a straight line whose cumulative count exceeds the threshold on the original image [Lines 220-222], and save the resulting image after drawing the straight line [Lines 224-226].

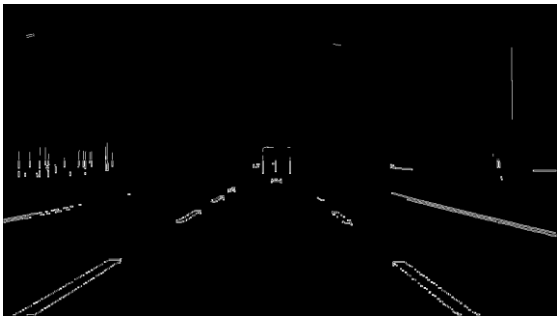
- Paste 3 input images and 9 output images.

img1.png

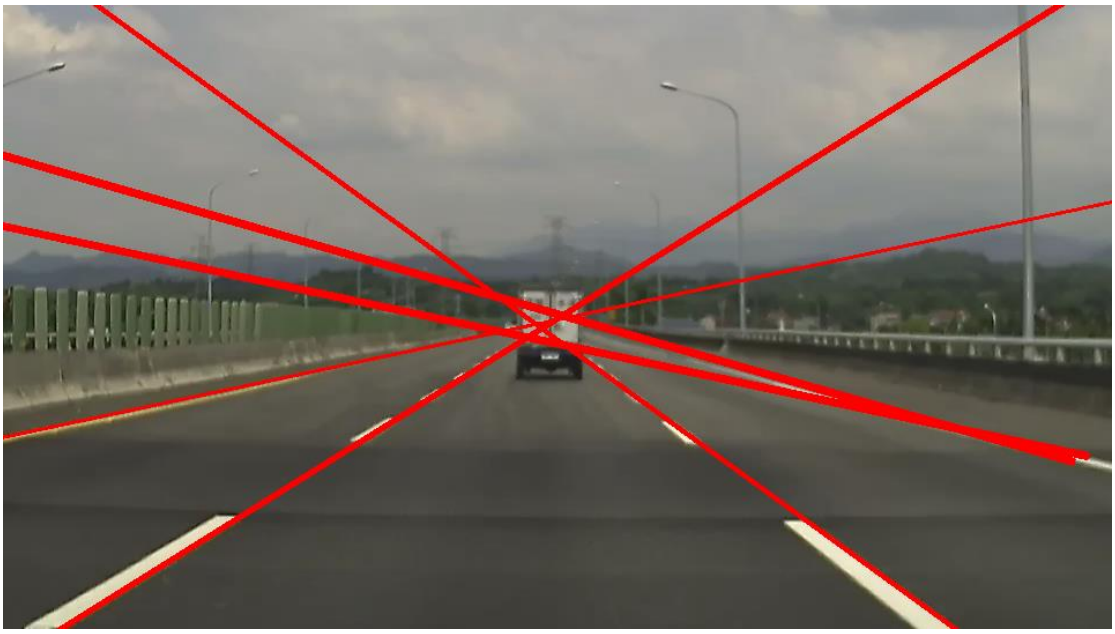
img2.png

img3.png

img1_q1.png

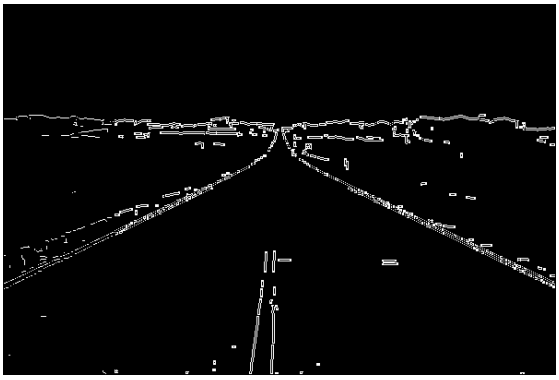


img1_q2.png

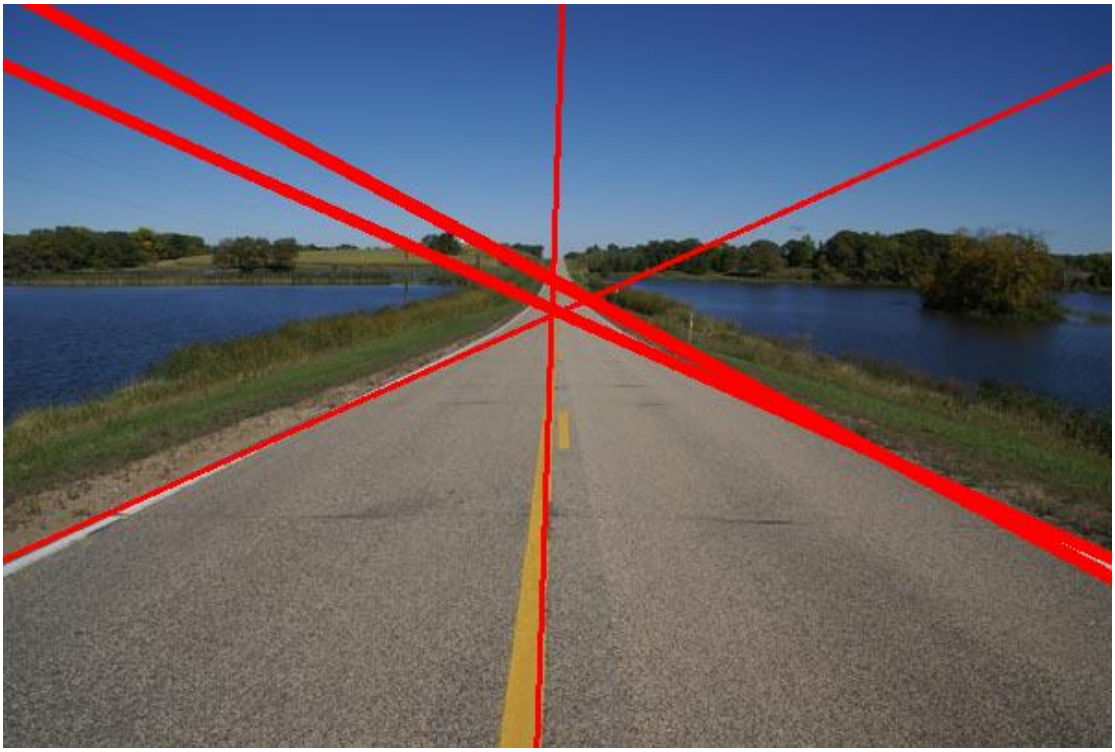


img1_q3.png

img2_q1.png



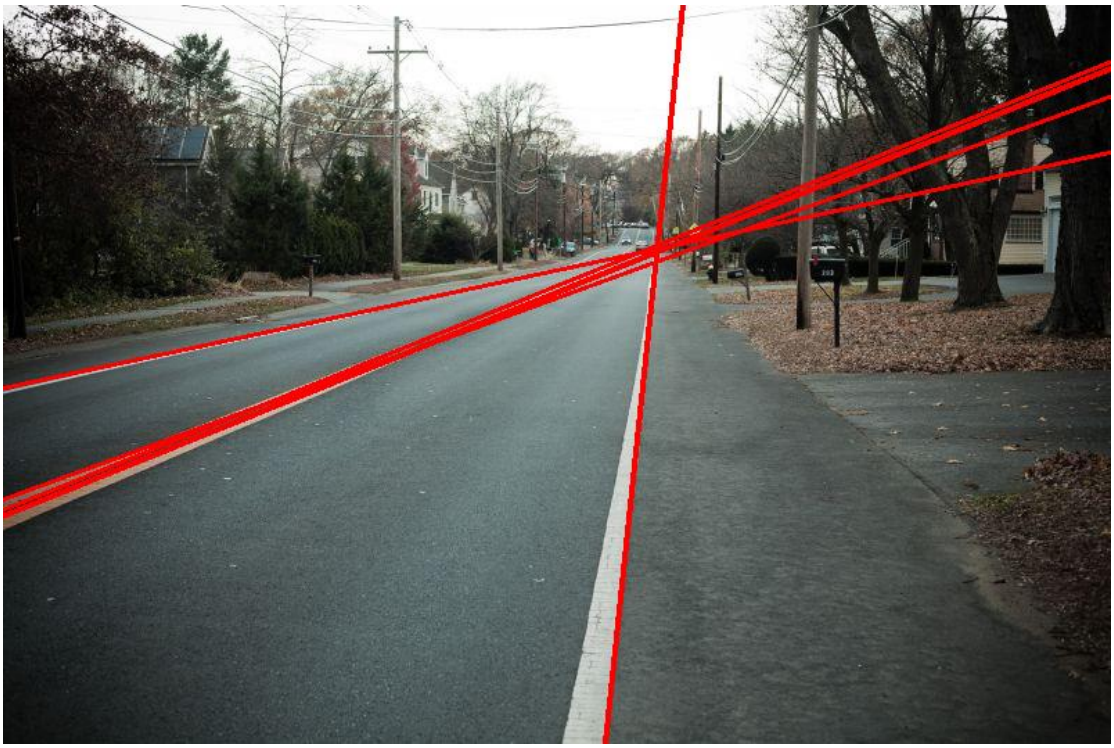
img2_q2.png



img2_q3.png

img3_q1.png

img3_q2.png



img3_q3.png

- Please state in your report the parameter setting that you choose for each input images.

1. **Setting of Gaussian filter parameter kernel_size.**

   First of all, the kernel_size of the Gaussian_filter function is all 5 because I want the image to be more blurry so that more noise can be filtered out. In fact, the lane line detection results of kernel_size=3 and kernel_size=5 are compared (other parameters are unchanged), indeed the detection effect of kernel_size=5 will be better. If kernel_size=3 is set, many straight lines that are not lane lines will be detected.

2. **Canny edge detection low_threshold and high_threshold settings.**

   Setting the low_threshold and high_threshold of gaussian_img1 and gaussian_img3 to 30 and 180 is the best way to display the edge. If the high_threshold of gaussian_img1 is lower than 180, it will cause the railing to be detected as a lane line. If it is higher than 180, it will cause the left one Yellow lane lines cannot be detected, and if the high_threshold of gaussian_img3 is lower than 180, multiple lane lines in the middle will be detected. If it is lower than 180, the lane lines on the left will also not be detected. Therefore, the compromise is The high_threshold is set to 180.

   If the high_threshold of gaussian_img2 is set too high (more than 80), the yellow lane line in the middle cannot be detected.

**3. Settings of hough_transform function parameters theta_res, rho_res, theta_min, theta_max, and mask_top.**

If the theta_res of canny_img1 and canny_img2 is less than 4, multiple lane lines on the right side will be detected, but if it is greater than 4, the lane lines on the left and right sides of canny_img1 will not be detected. If the theta_res of canny_img1 and canny_img2 is less than 4, multiple lane lines on the right side will be detected, but if it is greater than 4, the lane lines on the left and right sides of canny_img1 will not be detected. The same is true for canny_img3. If theta_res is less than 2, multiple duplicate lane lines will be detected.

The reason why rho_res is set to 1 is because as long as it is greater than 1, a large number of lines will be detected in the results of the three images (even lines that are not lane lines).

The settings for theta_min and theta_max are based on the approximate range of the lane lines in the image. Notably, both canny_img2 and canny_img3 have a theta_min of -90 because if it were set to a positive value, the right lane line would not be drawn. In summary, it took a long time to fine-tune these parameters to achieve what I consider the best result.

Regarding the mask_top parameter (whether to mask the upper half of the image), canny_img3 does not apply masking, while canny_img2 masks the upper 1/4 portion (handled specifically at line 217). This is because masking the entire upper half would result in both the left and right lane lines not being drawn.