MP1 Report

Course Number: ECE484
Team Name: Carla McCarface
Team Member:
Peng (penghan2)
Amith (amithr3)
Tracy (tracymt2)
Taylor (tyxia2)

Problem 4:

Results:

1) Salt-and-pepper noise using median filter

2) Salt-and-pepper noise using gaussian filter

3) Gaussian noise using gaussian filter

4) Gaussian noise using median filter

Problem 4

Median filter is better for filting out salt-and-pepper noise.

Gaussian filter is better for Gaussian (White) noise.

Reasons:

Salt-and-pepper noise contains random occurences of black and white pixels, which means "0" or "255".

We just want to fix this point but conserves the outlies.

So Median filter is more suitable because it replace the pixel with the median values of its neighbor pixels.

But Gaussian filter blurs features and smooths out all the changes which changes the outlies.

Gaussian (White) noise contains random pixels with random intensity values, which means anything between "0" and "255".

So Gaussian filter is more suitable because it smooths out all the changes.

Problem 5:

The most interesting part is choosing source points and destination points for perspective transform. To detect lanes better, we would like to choose four source points that can form a trapezoid that contains the left and right lane as shown in Figure 1. So that not only the lanes can show in image, but also the two sides trapezoid are parallel to lanes respectively. The two lower points are at the bottom of the image and the upper points are at the position that the two lanes are not too close to each other. Then we choose 4 destination points to form a square so that the lanes will nearly show in the middle of the image.

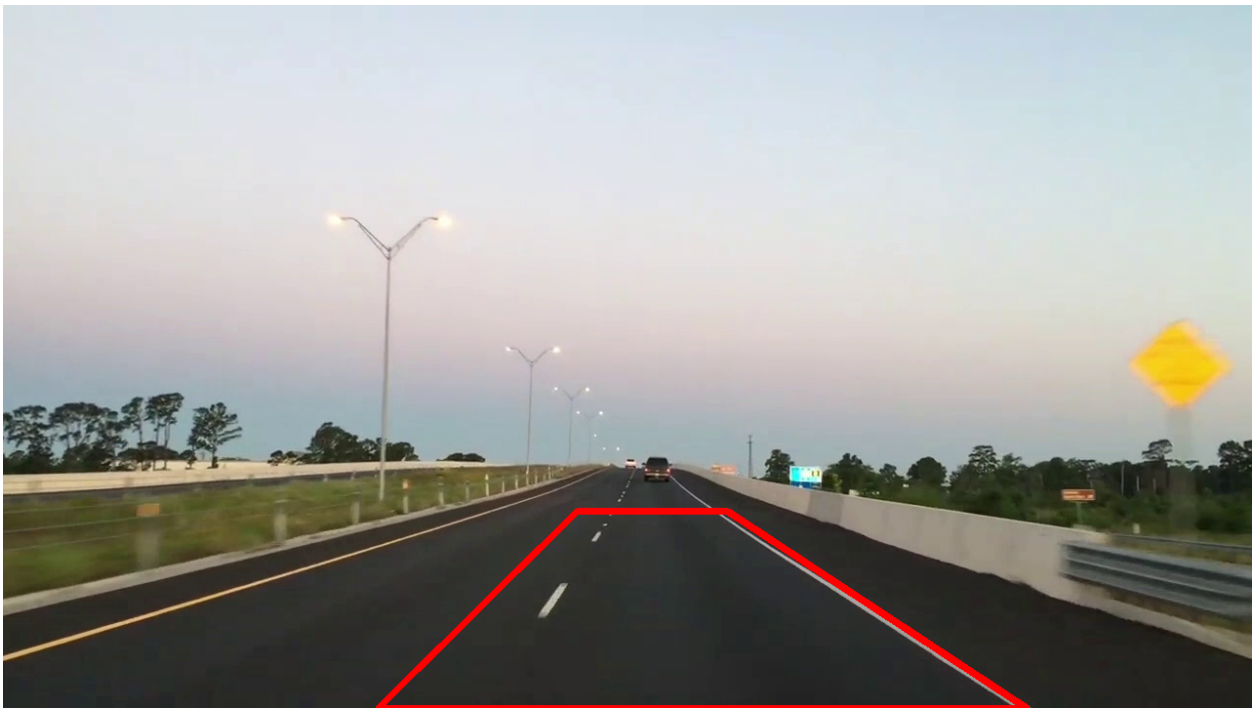

Figure 1. Four source points chosen to form a trapezoid.

Problem 6:

First, we modify the ordinates of source points and destination points.

Gazebo:
source points = np.float32([[265, 260], [15, 370], [610, 370], [365, 260])
destination points = np.float32([[0, 0], [0, 300], [300, 300], [300, 0]])

Video:
source points ([[500, 250], [270, 370], [860, 370], [730, 250]])
destination points = np.float32([[0, 0], [0, 300], [300, 300], [300, 0]])

To detect lanes better, we would like to choose four source points that can form a trapezoid that contains the left and right lane. However, the images from Gazebo and videos have different

sizes. Also, lanes in images from Gazebo and videos have different positions. So, the ordinates of source points and destination points in the two scenarios need to be modified based on the rule described above.

Second, we change the size of windows that are used to fit the lane. The reason is that in the video scenario, we have white arrows on the ground, which affects the detection of white dash lanes. So, we change the ordinates of left and right bases as listed below to make the windows not contain the arrows in the middle of two lanes.

Gazebo:
leftx_base = np.argmax(histogram[30:midpoint-50]) + 30
rightx_base = np.argmax(histogram[midpoint+50:-50]) + midpoint +50

Video:
leftx_base = np.argmax(histogram[30:midpoint]) + 30
rightx_base = np.argmax(histogram[midpoint:-50]) + midpoint

Problem 7

Link:
GEM scenario:
https://www.youtube.com/watch?v=A-sk3nB55Mo
Video_0011_scenario:
https://www.youtube.com/watch?v=J_RaZub5k7I
Video_0055_scenario:
https://www.youtube.com/watch?v=YswOaoslEek

Problem 8

Our lane detector does not work in this snowfall condition. We think there are some reasons for this issue.
First, the brightness of snow is too high. So, the “color_thersh” function we use would pass the color of snow, so that the snow would show in our final combined image and the white lanes cannot be easily followed.
Second, snow makes the edge not that obvious. So, the “gradient thersh” function we use cannot correctly detect the edge of lane, which causes the failure of lane detection.