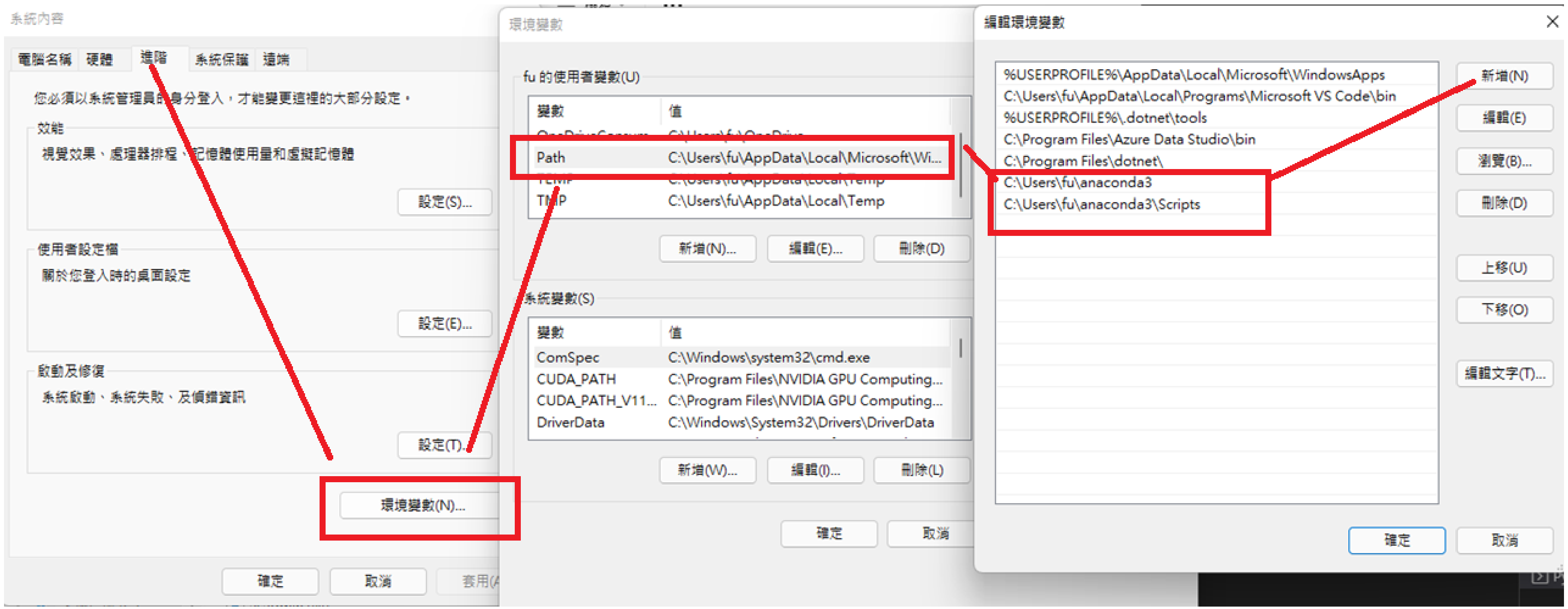
PERCEPTION FOR AUTONOMOUS CARS 2

2024/03/12

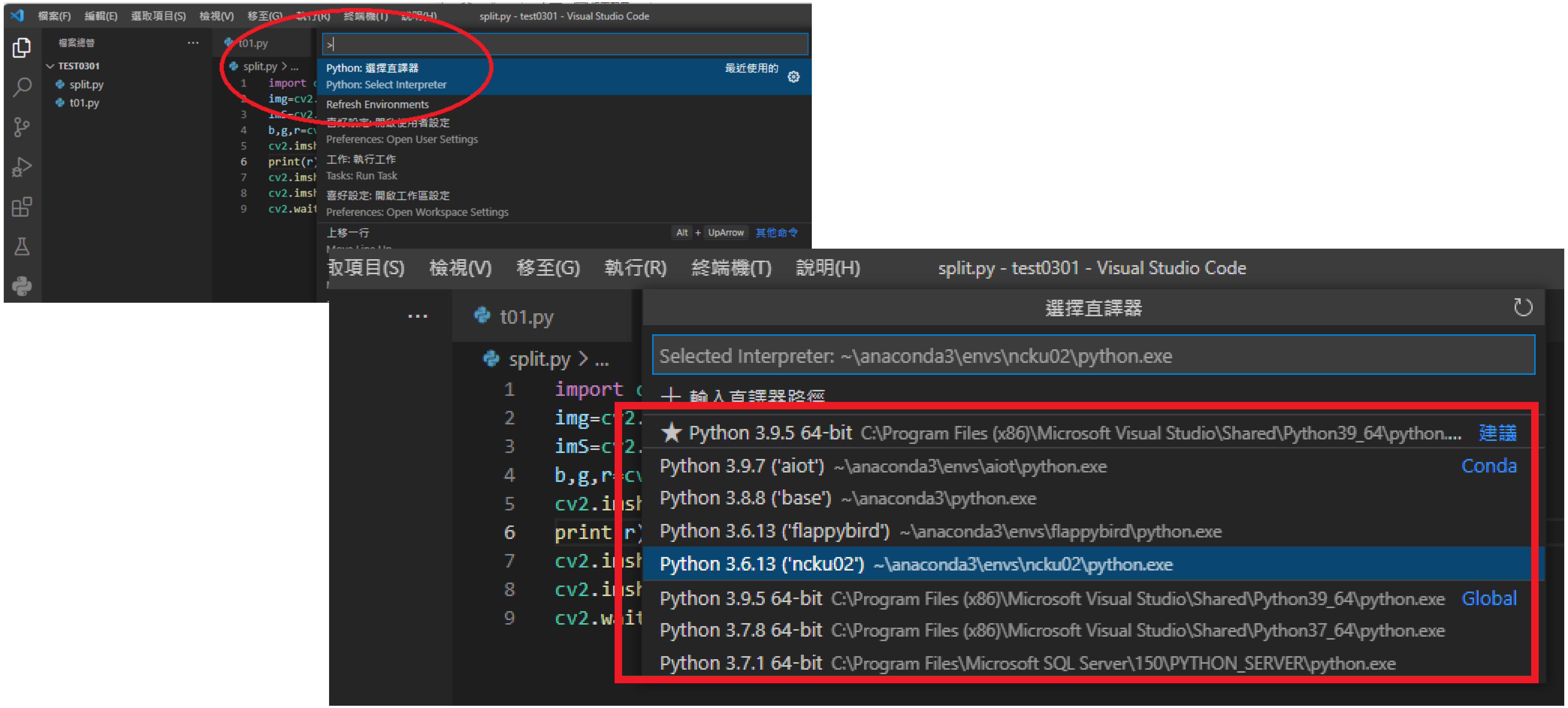
Using Python with Anaconda and Visual Studio Code (Windows)

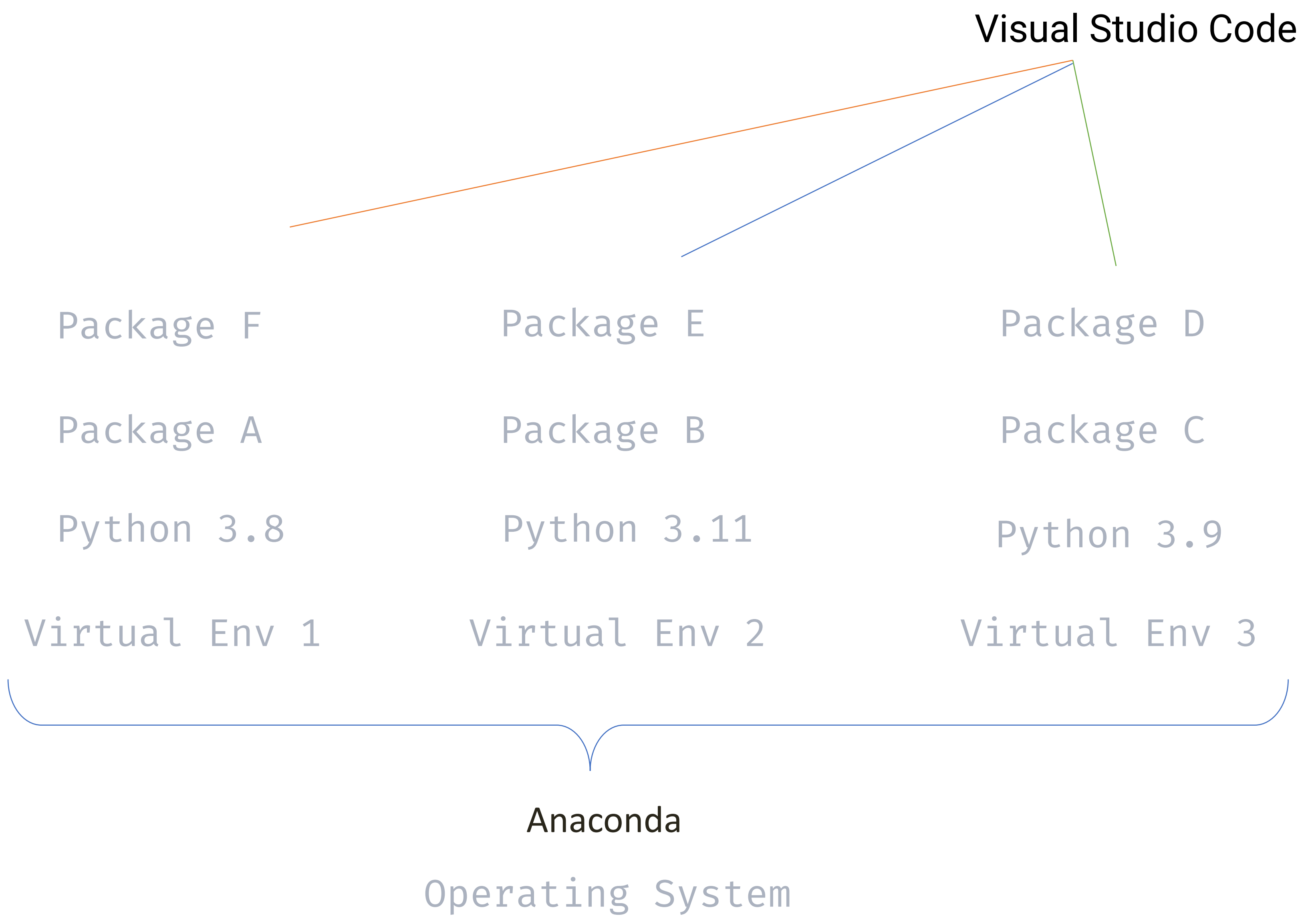
1. Press Win+X keys on the keyboard and select System from the list.
2. Inside the System window, click on Advanced system settings located at the left pane.



Using Python with Anaconda and Visual Studio Code (Windows)

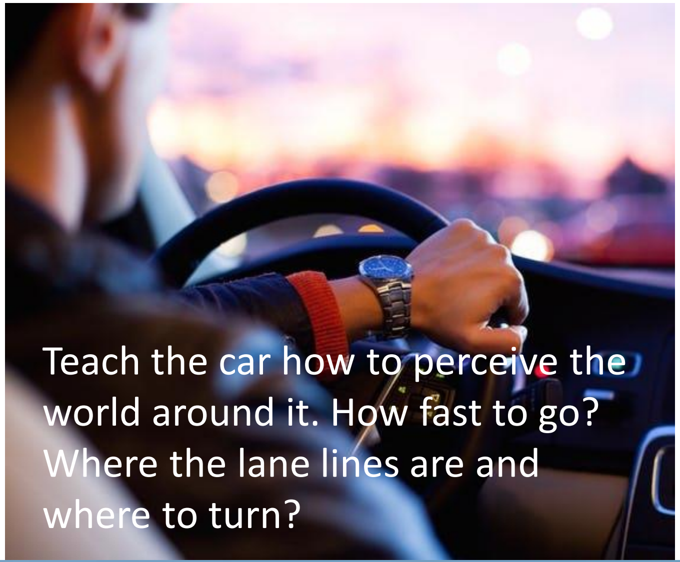
- Visual Studio Code → Ctrl + shift + P key

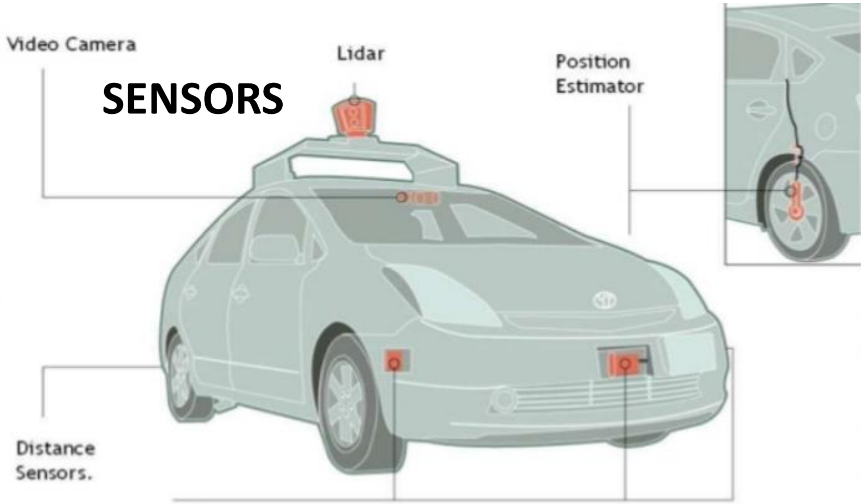



Visual Studio Code
Package F
Package E
Package D
Package A
Package B
Package C
Python 3.8
Python 3.11
Python 3.9
Virtual Env 1
Virtual Env 2
Virtual Env 3
Anaconda
Operating System

Finding Lane Lines

Finding Lane Lines on the Road





Finding Lane Lines on the Road

- ☐ Color
- ☐ Shape
- ☐ Orientation
- ☐ Position in the image

Finding Lane Lines on the Road

☑ Color
☑ Shape
☑ Orientation
☑ Position in the image

Identifying the lane lines using Color

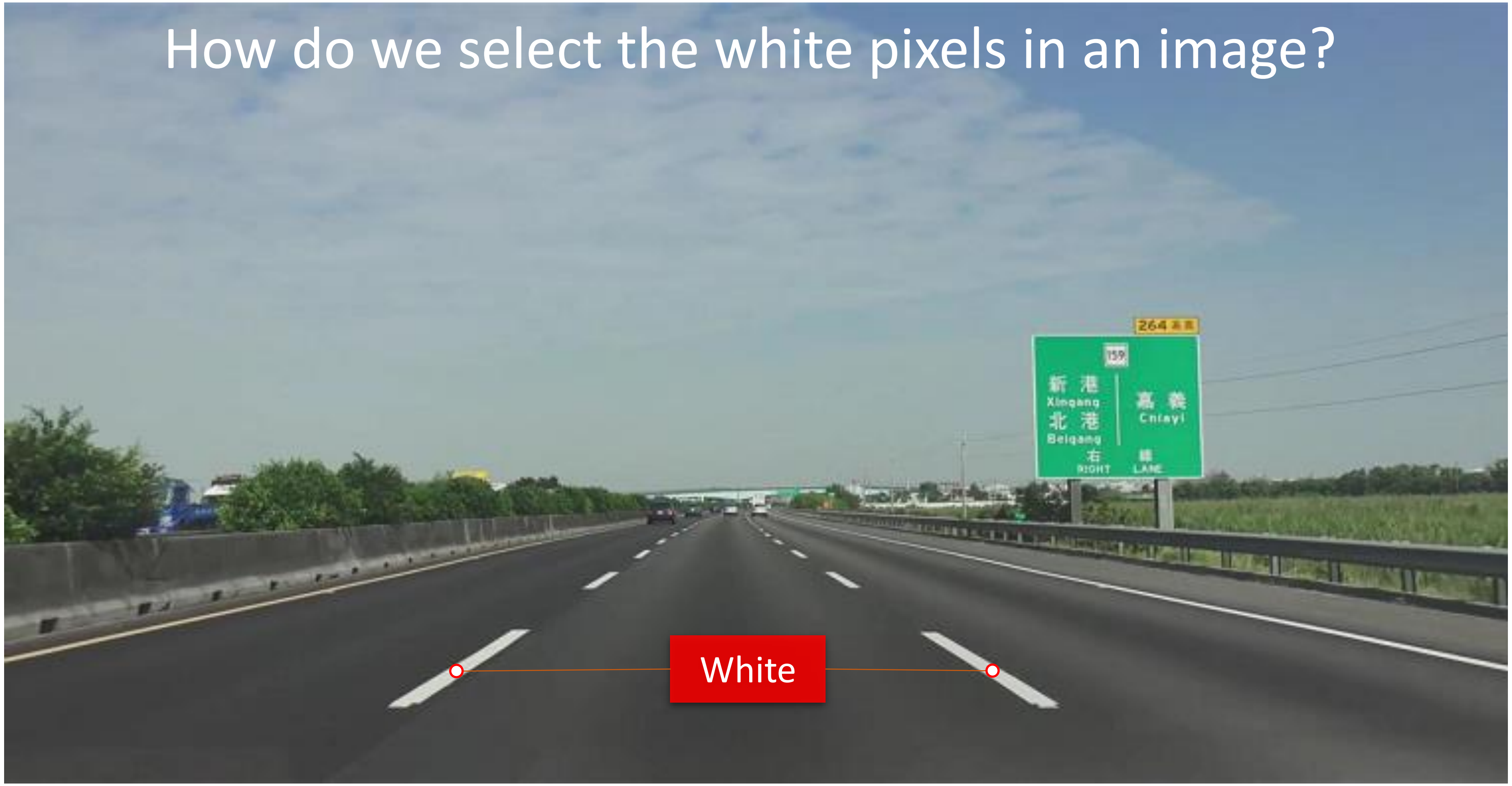
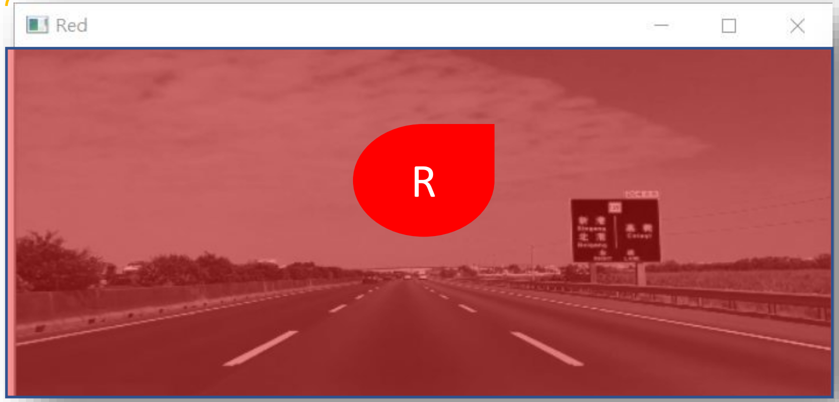
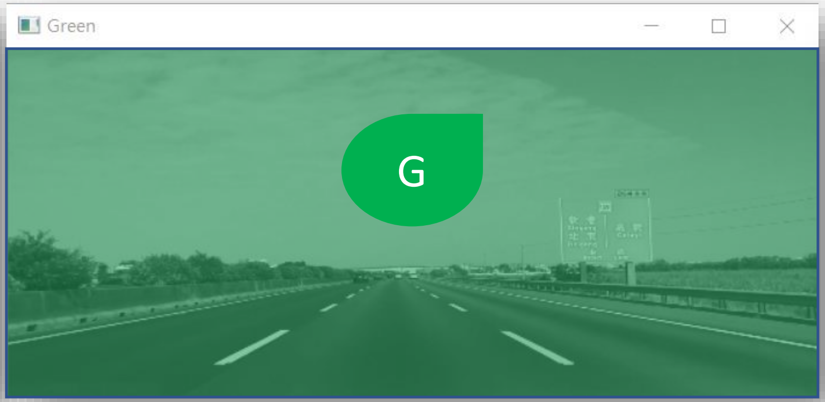
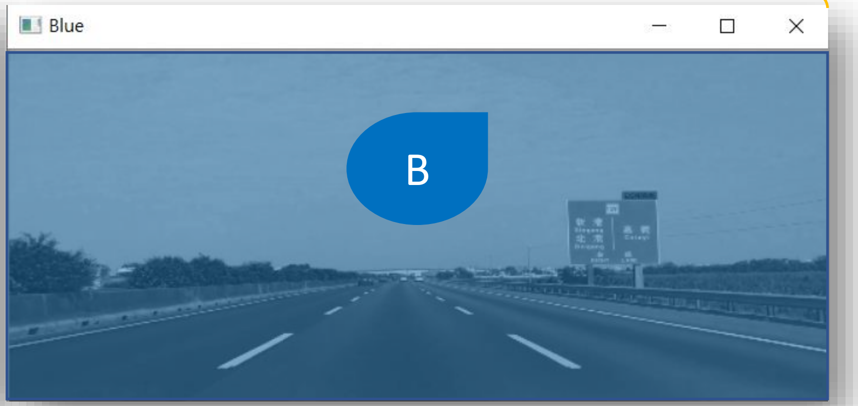


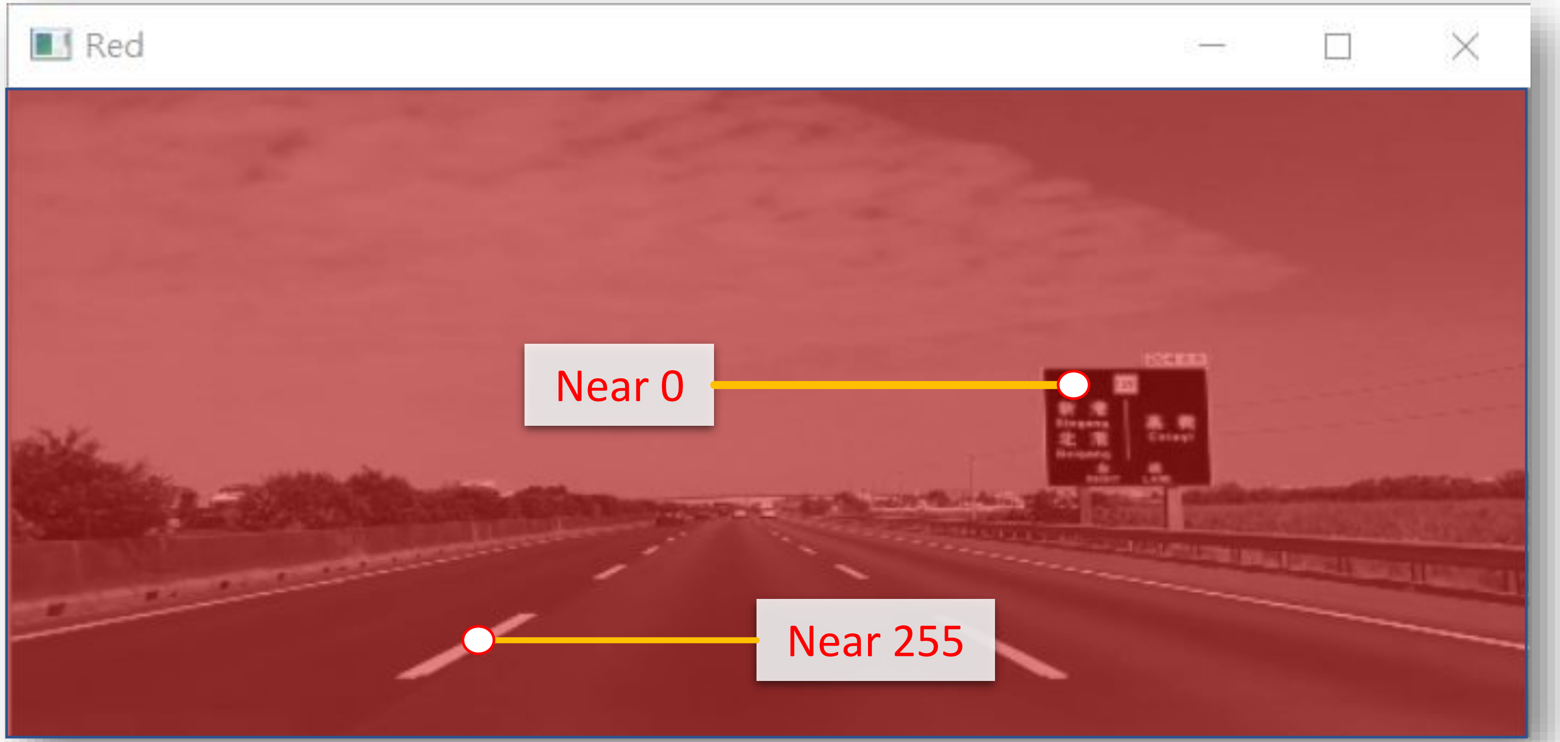
Image is actually made up of a stack of three images

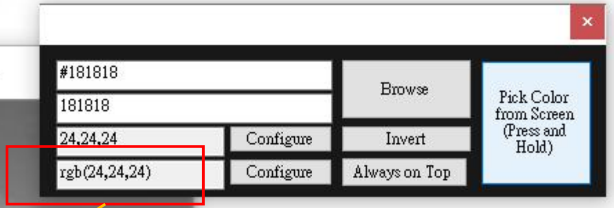
```
import cv2
img=cv2.imread('c:\\ncku\\test01\\images02\\Color_Selection.jpg',cv2.IMREAD_COLOR)
imS=cv2.resize(img,(560,240))
b,g,r=cv2.split(imS)
cv2.imshow("Red",r)
cv2.imshow("Green",g)
cv2.imshow("Blue",b)
cv2.waitKey(0)
```



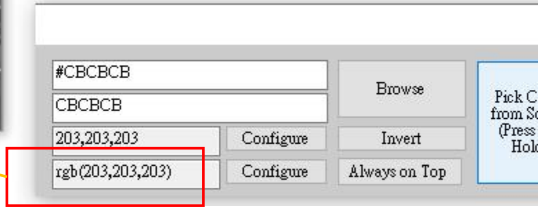




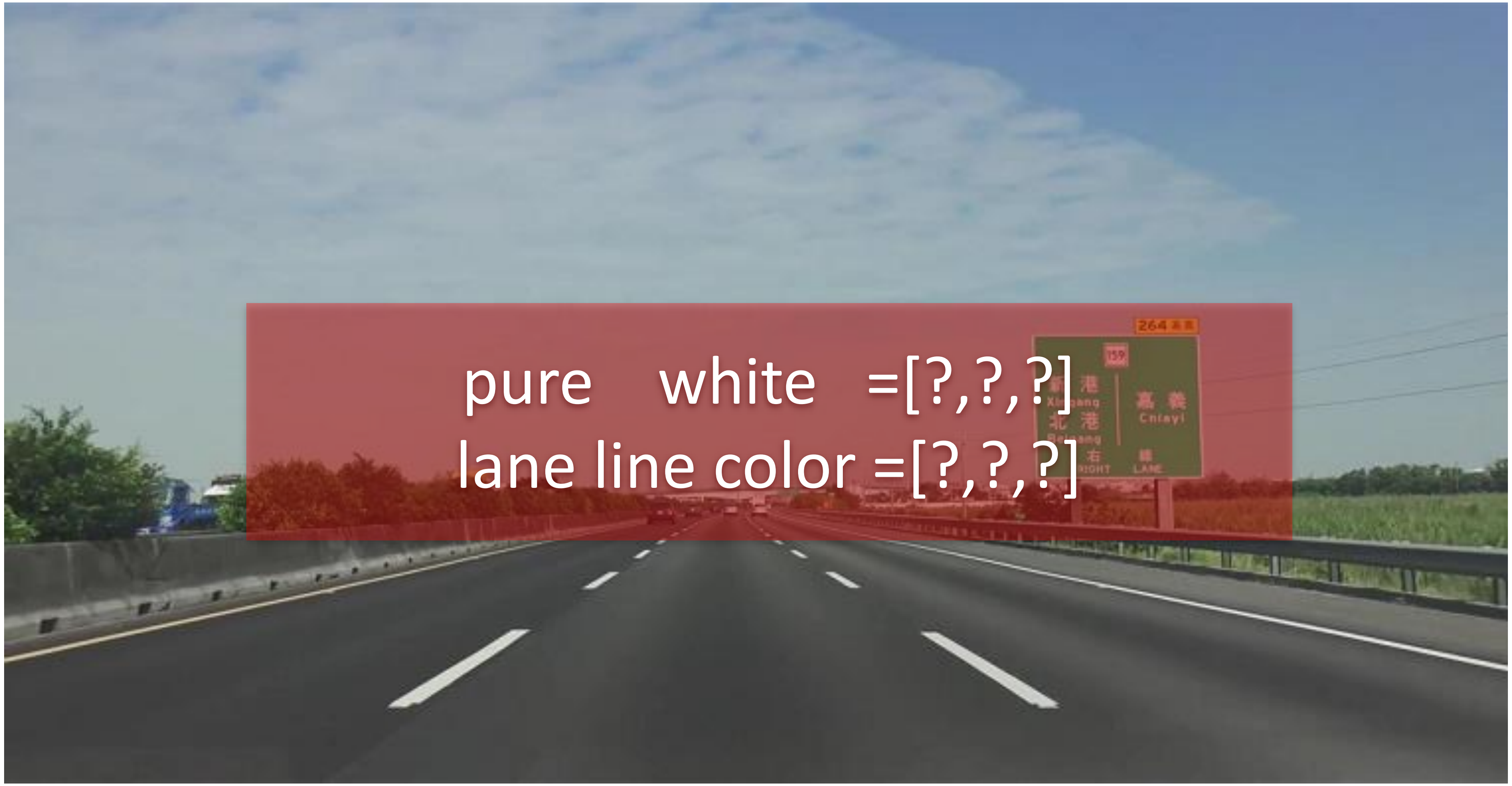




Color Picker download URL:
https://sourceforge.net/projects/colorfinder/



R G B 3 Color Chanel (0~255)


pure white =[?,?,?]
lane line color =[?,?,?]

Quiz Question

What color is pure white in our combined red + green + blue [R, G, B] image?

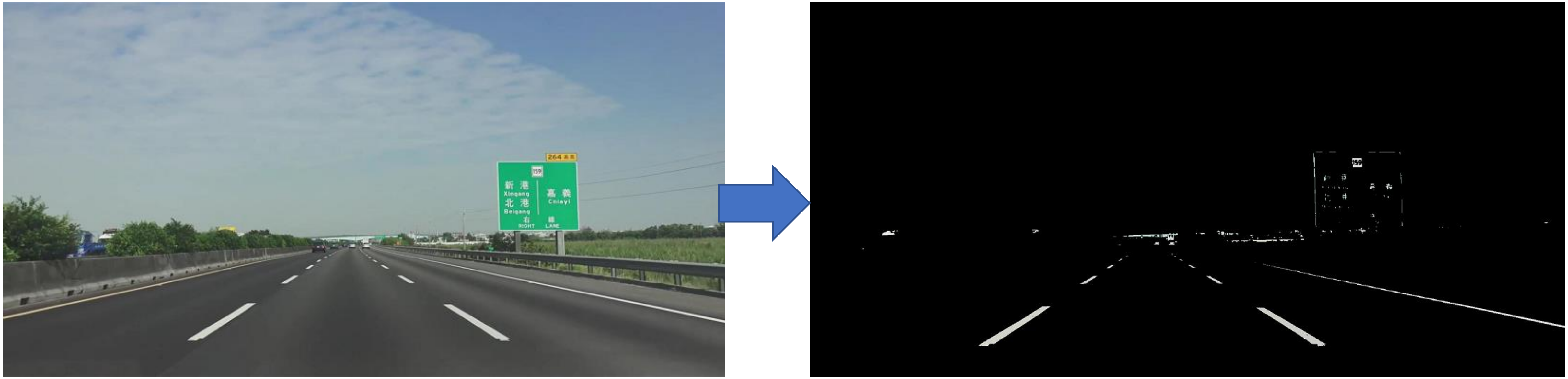
- [0,0,0]
- [0,255,255]
- [100,150,200]
- [255,255,255]

Quiz Question

What color is pure white in our combined red + green + blue [R, G, B] image?

- ❍ [0,0,0]
- ❍ [0,255,255]
- ❍ [100,150,200]
- ● [255,255,255]

Color Selection



The original image (left), and color selection applied (right).

Color Selection Code Example


```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np

#Read in the image
image = mpimg.imread('test.jpg')

#Grab the x and y size and make a copy of the image
ysize = image.shape[0]
xsize = image.shape[1]
color_select = np.copy(image)

#Define color selection criteria
###### MODIFY THESE VARIABLES TO MAKE YOUR COLOR SELECTION
blue_threshold = 0
green_threshold = 0
red_threshold = 0
######

rgb_threshold = [blue_threshold, green_threshold, red_threshold]
```

Code: 01Color_Selection.txt

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np

# Read in the image
image = mpimg.imread('test01.jpg')
```

```
# Grab the x and y size and make a copy of the image
ysize = image.shape[0]
xsize = image.shape[1]
print(ysize)
print(xsize)
```

```
color_select = np.copy(image)
# Define color selection criteria
###### MODIFY THESE VARIABLES TO MAKE YOUR COLOR SELECTION
blue_threshold = 0
green_threshold = 0
red_threshold = 0
######
rgb_threshold = [blue_threshold, green_threshold, red_threshold]
```

image[:,:,0] < rgb_threshold[0]→True or False

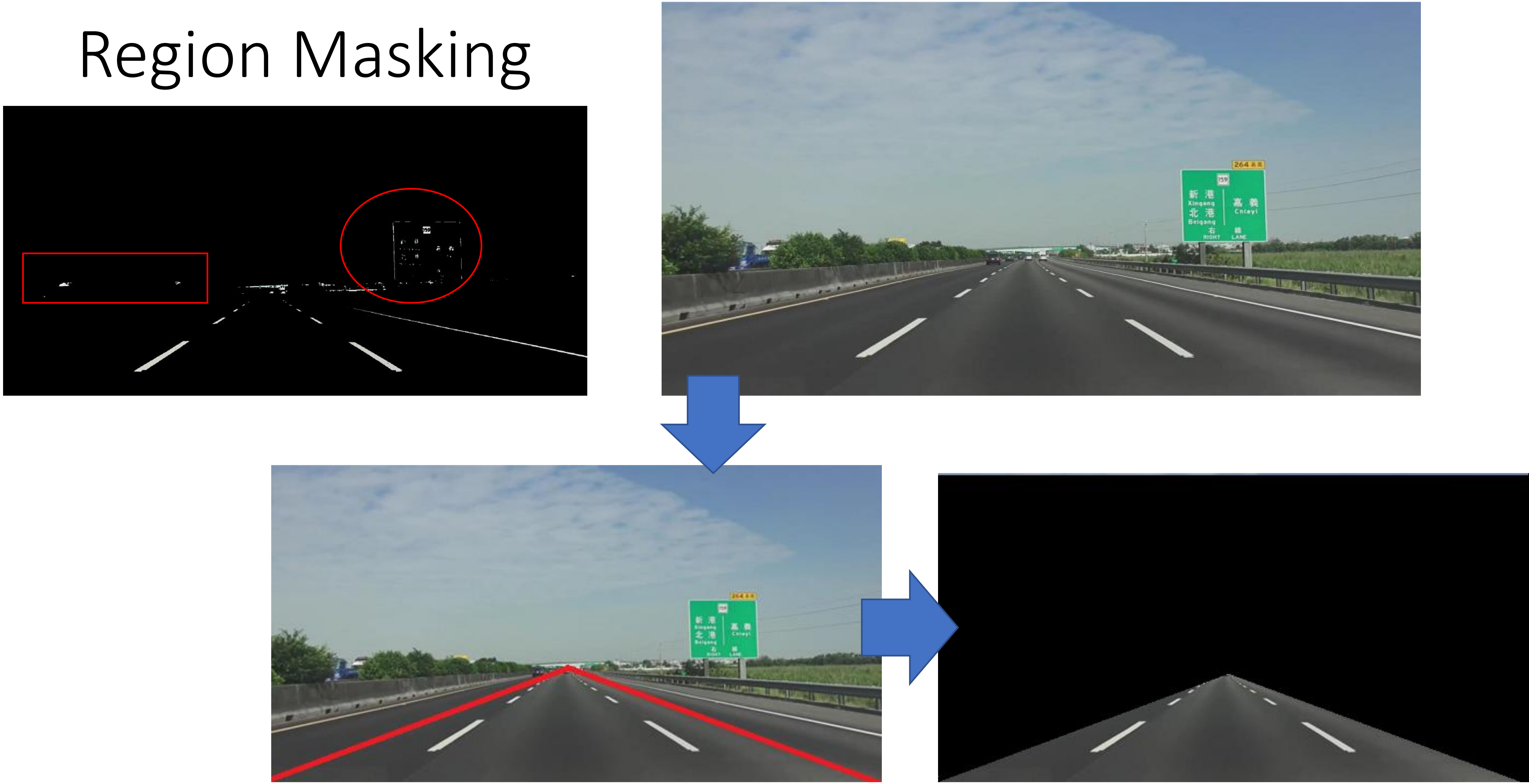
```
# Do a boolean or with the "|" character to identify
# pixels below the thresholds
thresholds = (image[:,:,0] < rgb_threshold[0]) \
            | (image[:,:,1] < rgb_threshold[1]) \
            | (image[:,:,2] < rgb_threshold[2])
color_select[thresholds] = [0,0,0]

# Display the image
plt.imshow(color_select)
```

```
imgB = image[:,:,0]      #0 Channel
plt.imshow(imgB)
```

```
R = np.copy(image)   # copy image into new array
R[:,:,1]=0           # set green channel to 0
R[:,:,2]=0           # set Red channel to 0
plt.imshow(R)
plt.show()           # display new image
```

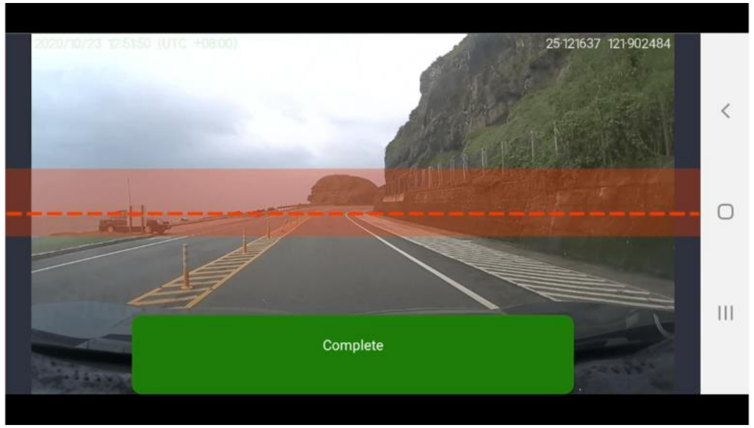
document: [Coding_up_a_Color_Selection.docx]

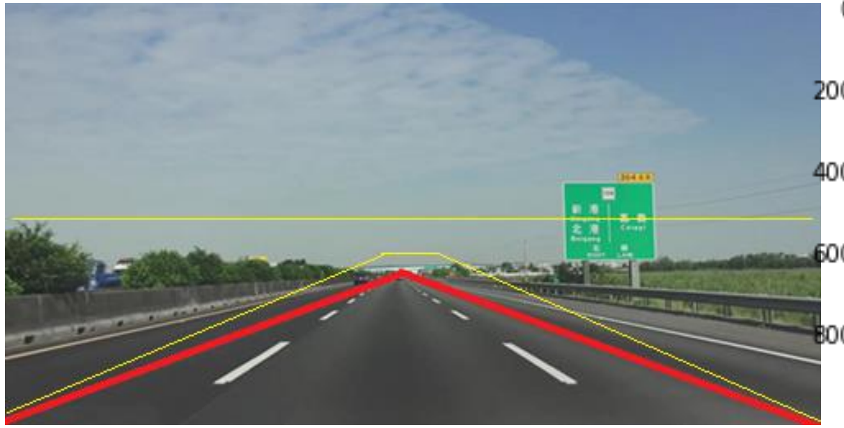
Region Masking

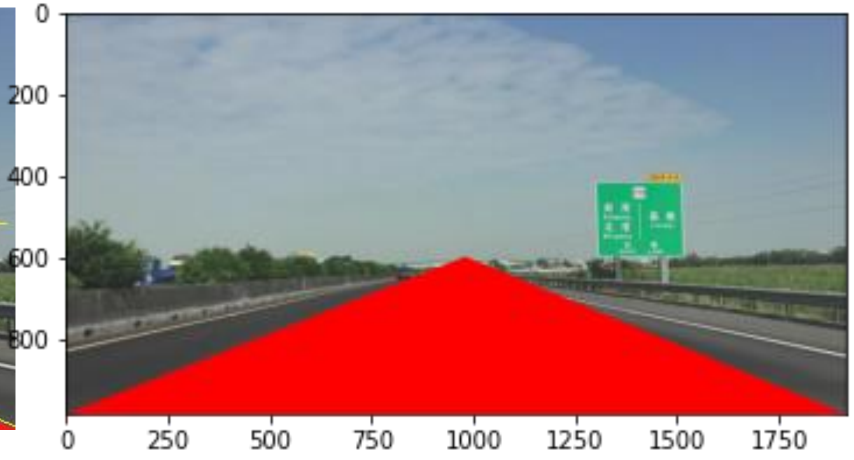


Coding up a Region of Interest Mask

VIA Mobile360 D700 AI Dash Cam - ADAS (viatech.com)







Code: 02Region_Masking.txt

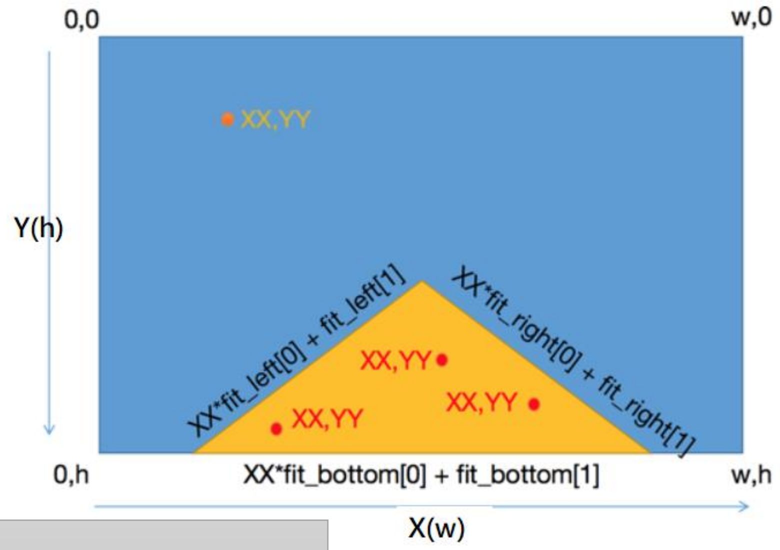
```
# Define a triangle region of interest.
#Keep in mind the origin (x=0, y=0) is in the upper
# left in image processing
left_bottom = [0, 539]
right_bottom = [900, 300]
apex = [400, 0]
```

numpy.polyfit

`numpy.polyfit(x, y, deg, rcond=None, full=False, w=None, cov=False)`

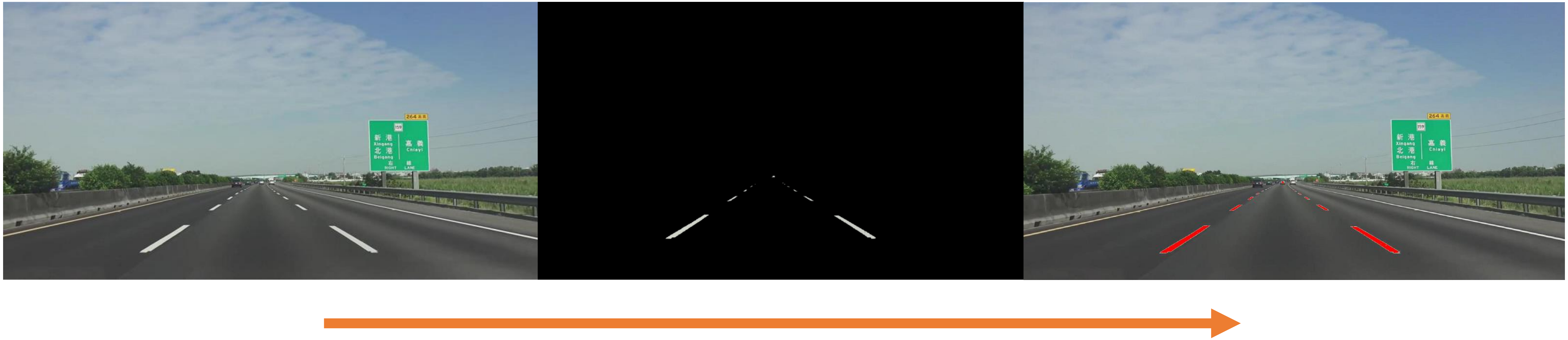
```
# Fit lines (y=Ax+B) to identify the  3 sided region of interest
# np.polyfit() returns the coefficients [A, B] of the fit
fit_left = np.polyfit((left_bottom[0], apex[0]), (left_bottom[1], apex[1]), 1)
fit_right = np.polyfit((right_bottom[0], apex[0]), (right_bottom[1], apex[1]), 1)
fit_bottom = np.polyfit((left_bottom[0], right_bottom[0]), (left_bottom[1],right_bottom[1]), 1)
```

```
# Find the region inside the lines
XX, YY = np.meshgrid(np.arange(0, xsize), np.arange(0, ysize))
region_thresholds = (YY > (XX*fit_left[0] + fit_left[1])) & \
                    (YY > (XX*fit_right[0] + fit_right[1])) & \
                    (YY < (XX*fit_bottom[0] + fit_bottom[1]))

# Color pixels red which are inside the region of interest
region_select[region_thresholds] = [255, 0, 0]
```




```
# Display the image
#check matplotlib.pyplot.plot (https://matplotlib.org/api/pyplot_api.html#module-matplotlib.pyplot)
#tip:plt.plot([0], [539], 'bo')
plt.imshow(region_select)
```

Color and Region Selection



Combining Color and Region Selections

- Now you've seen how to mask out a region of interest in an image. Next, let's combine the mask and color selection to pull only the lane lines out of the image.
- Check out the code below. Here we're doing both the color and region selection steps, requiring that a pixel meet both the mask and color selection requirements to be retained.

Code: 03Combining_Color_and_Region_Selections.txt

Reference: https://matplotlib.org/2.0.2/api/pyplot_api.html

Finding Lines of Any Color

What is Computer Vision?

Canny Edge Detection

- It was developed by John F. Canny in 1986.
- With edge detection, the goal is to identify the boundaries of an object in an image.
- First, Convert to grayscale. And next, compute the gradient.

```
edges = cv2.Canny(gray, low_threshold, high_threshold)
```

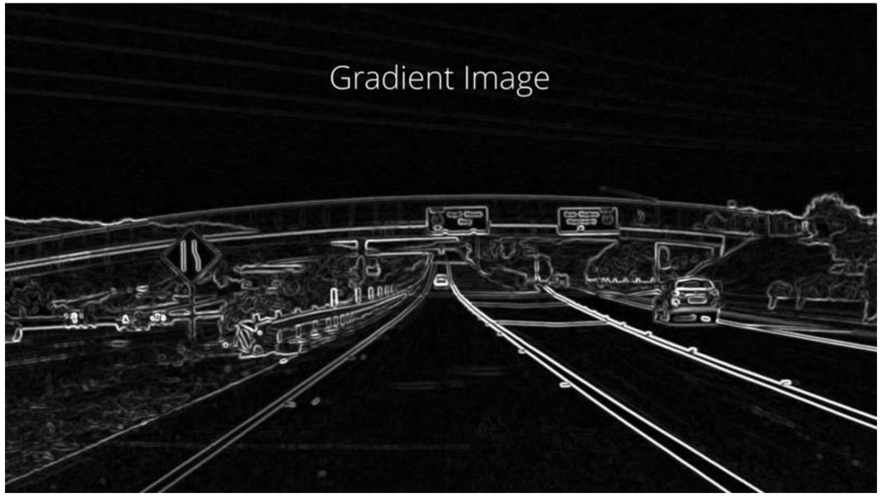
John F. Canny

John Canny in his office at University of California, Berkeley (2013)

$$f(x,y) = pixel\ value$$

Canny and gradient

$$\frac{dy}{dx} = \Delta(pixel\ value)$$



https://en.wikipedia.org/wiki/Canny_edge_detector

Canny Edge Detection

The line in the plot right shows where I took a cross section through the image. The wiggles in the blue line indicate changes in intensity along that cross section through the image. Check all the boxes of the letters along this cross section, where you expect to find strong edges.

- ☐ A
- ☐ B
- ☐ C
- ☐ D
- ☐ E
- ☐ F

Canny Edge Detection

The line in the plot right shows where I took a cross section through the image. The wiggles in the blue line indicate changes in intensity along that cross section through the image. Check all the boxes of the letters along this cross section, where you expect to find strong edges.

- ✓ A
- ☐ B
- ✓ C
- ☐ D
- ✓ E
- ☐ F

Canny Edges

The original image (left), and edge detection applied (right).

Canny Edge Detection in Action

First, we need to read in an image:

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
image = mpimg.imread('exit-ramp.jpg')
plt.imshow(image)
```

Let's go ahead and convert to grayscale.

```
import cv2  #bringing in OpenCV libraries
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
#grayscale conversion
plt.imshow(gray, cmap='gray')
```

```
edges = cv2.Canny(gray, low_threshold, high_threshold)
```

```python
#doing all the relevant imports
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import cv2
```

```python
# Read in the image and convert to grayscale
image = mpimg.imread('exit-ramp.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
```

```python
# Define a kernel size for Gaussian smoothing / blurring
# Note: this step is optional as cv2.Canny() applies a 5x5 Gaussian internally
kernel_size = 3
blur_gray = cv2.GaussianBlur(gray,(kernel_size, kernel_size), 0)
```

```python
# Define parameters for Canny and run it
# NOTE: if you try running this code you might want to change these!
low_threshold = 1
high_threshold = 10
edges = cv2.Canny(blur_gray, low_threshold, high_threshold)

# Display the image
plt.imshow(edges, cmap='Greys_r')
```

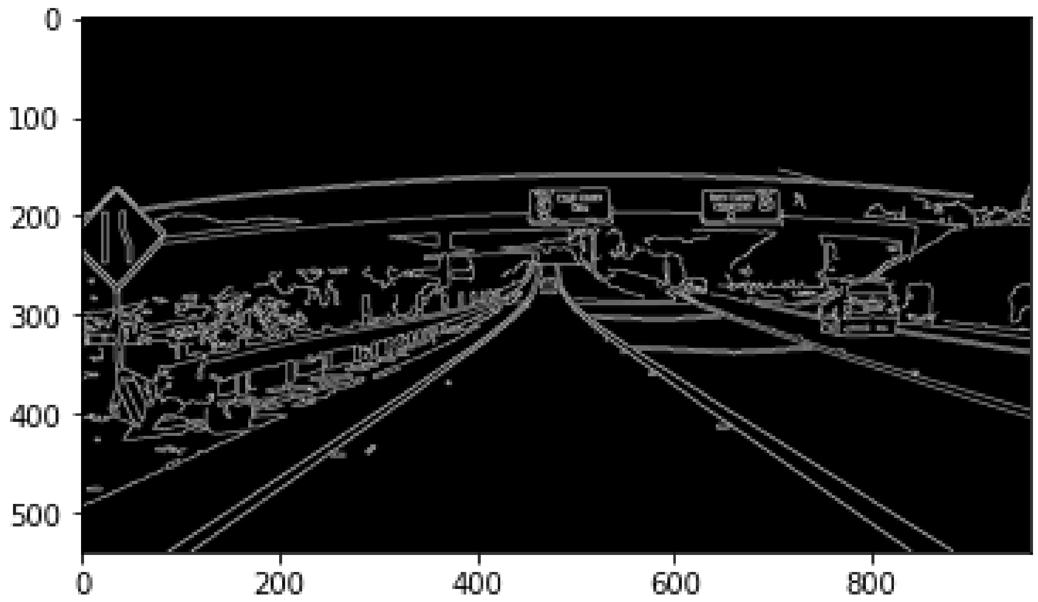
Code: [04Canny_to_Detect_Lane_Lines.txt]   [CannyDetect.docx]

Answer:

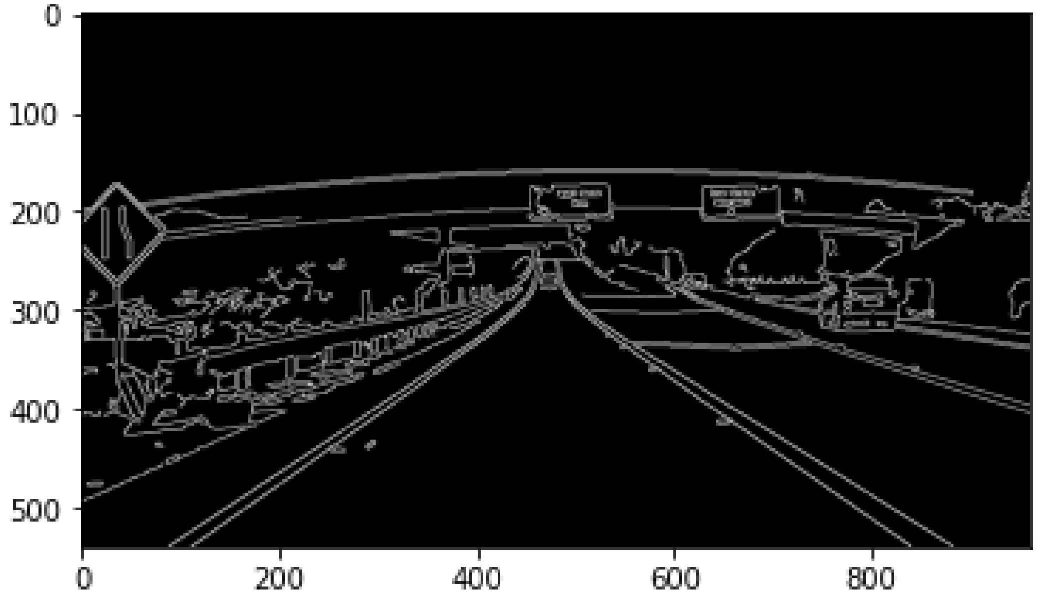
Kernel Size: 5 , Gaussian smoothing

50 < Threshold < 150

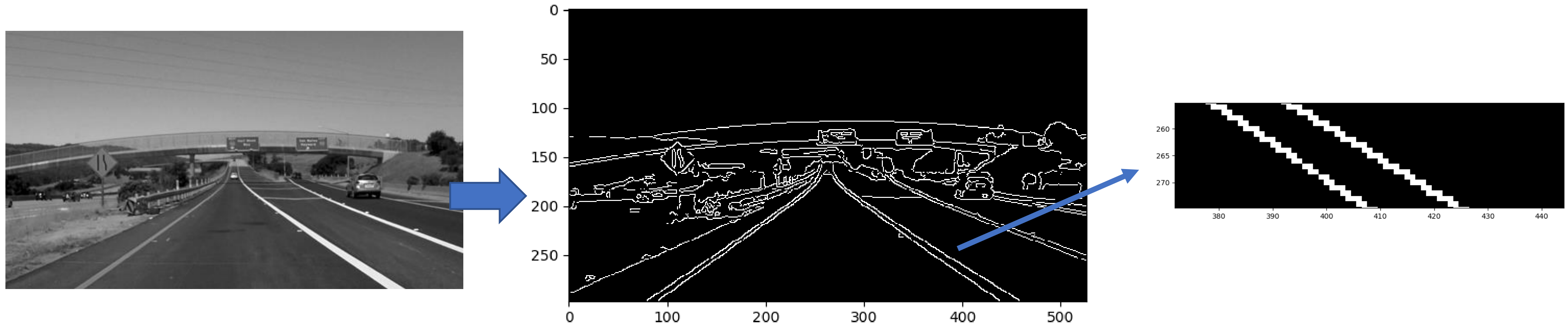
kernel Size : 3



kernel Size : 5



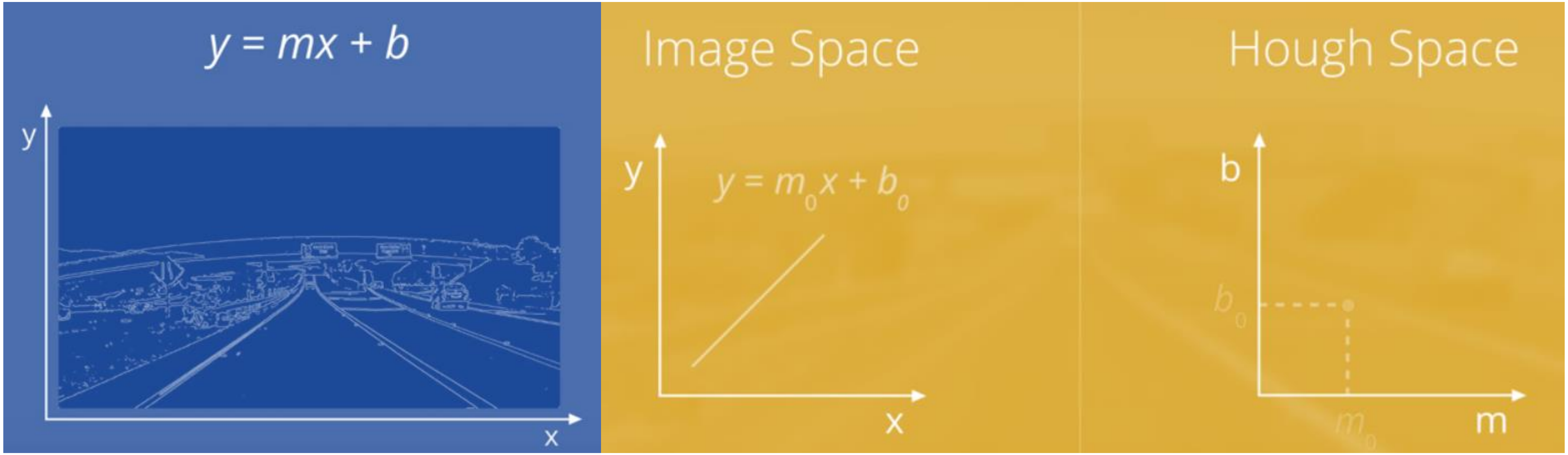
Let’s connect the dots.



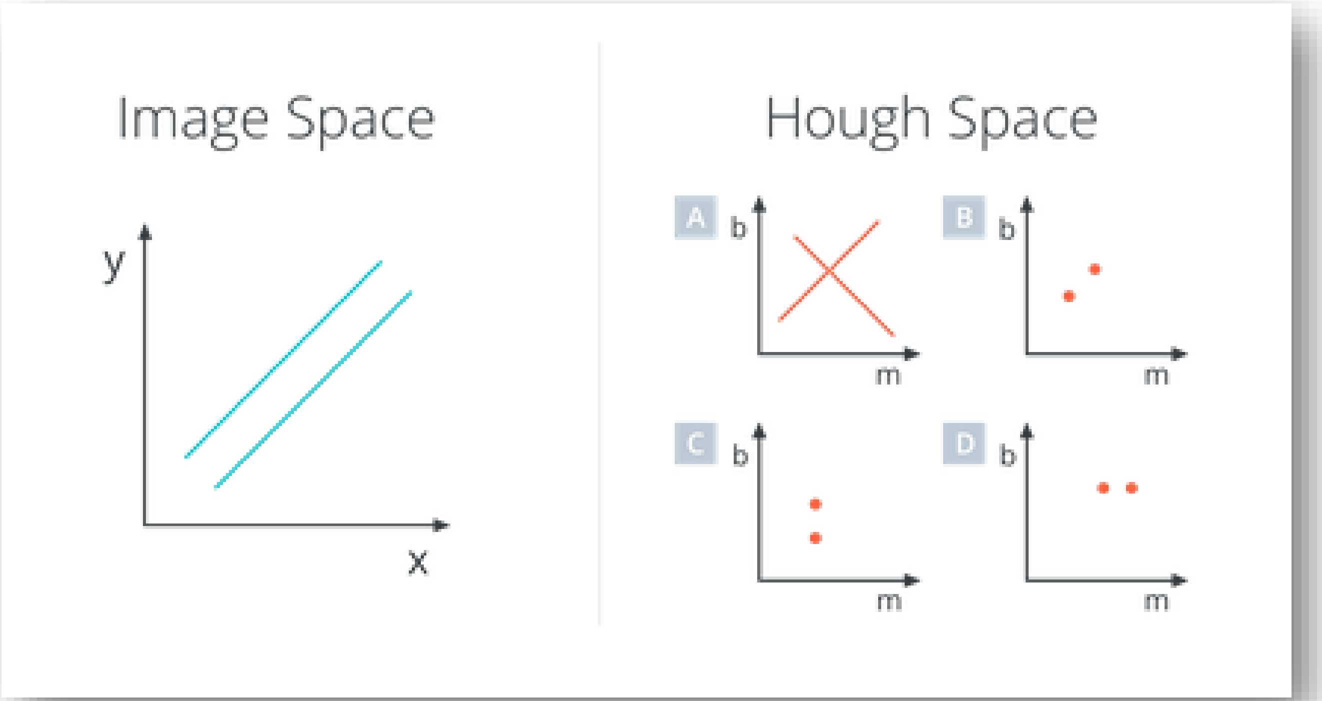
Hough Transform

https://en.wikipedia.org/wiki/Hough_transform

Using the Hough Transform to Find Lines from Canny Edges



Hough Transform



Question 1 of 5

What will be the representation in Hough space of two parallel lines in image space?

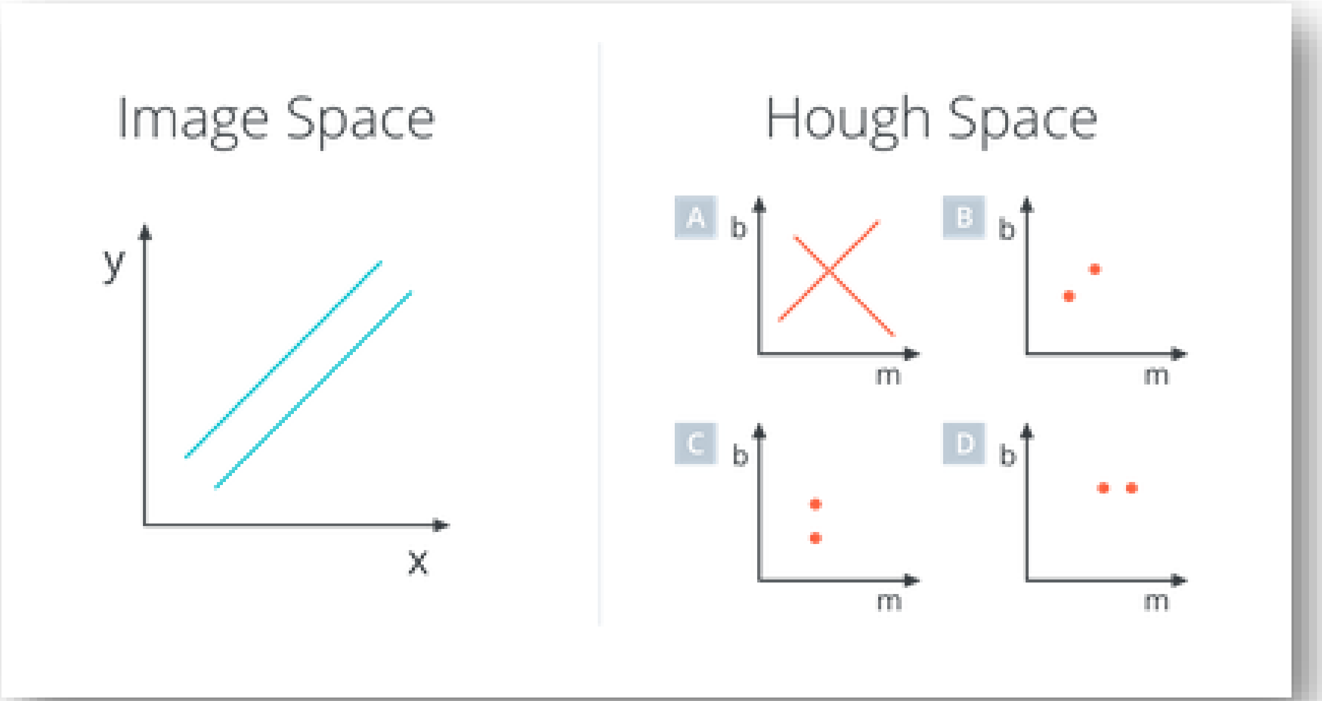
❍A

❍B

❍C

❍D

Hough Transform



Question 1 of 5

What will be the representation in Hough space of two parallel lines in image space?

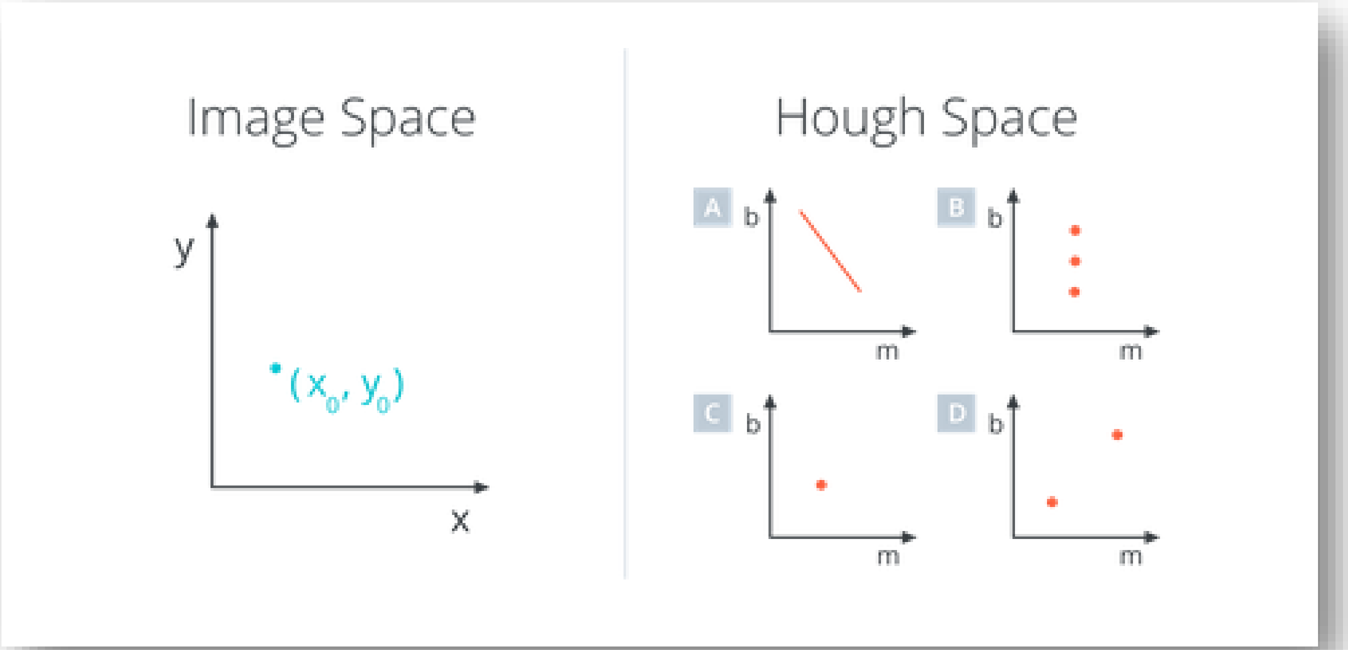
❍ A

❍ B

● C

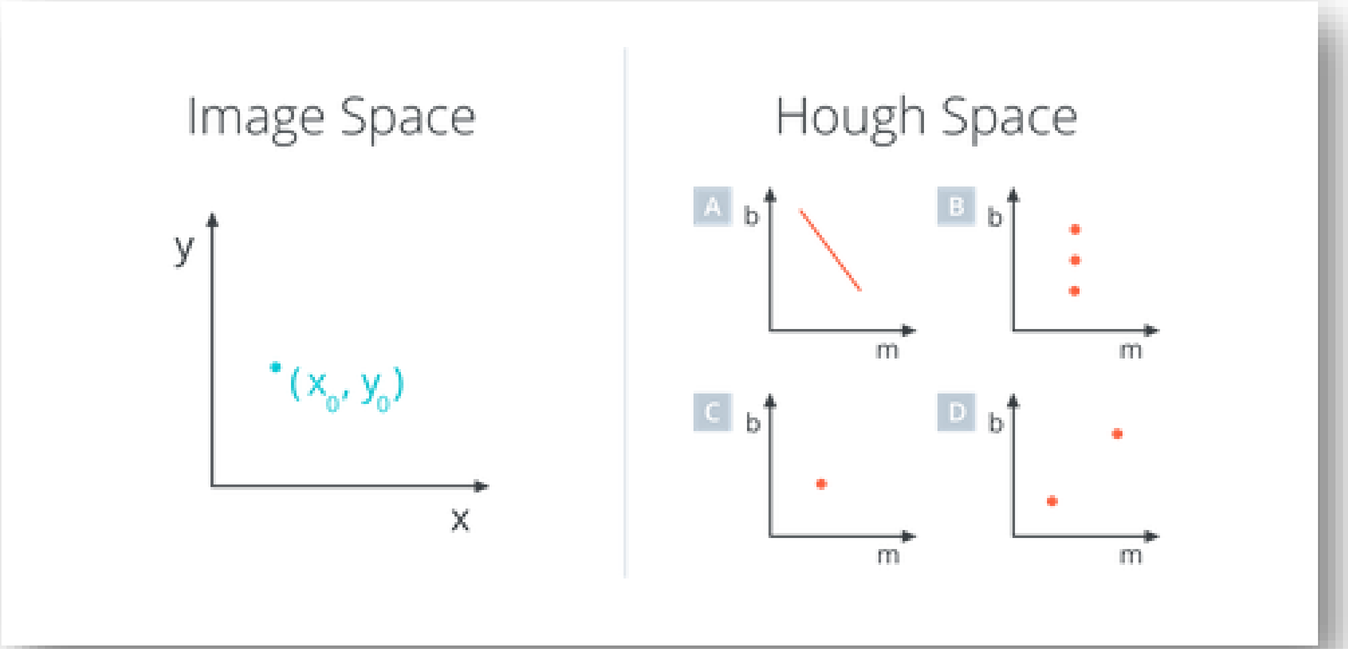
❍ D

Hough Transform



Question 2 of 5
What does a point in image space correspond to in Hough space?
- ❍A
- ❍B
- ❍C
- ❍D

Hough Transform

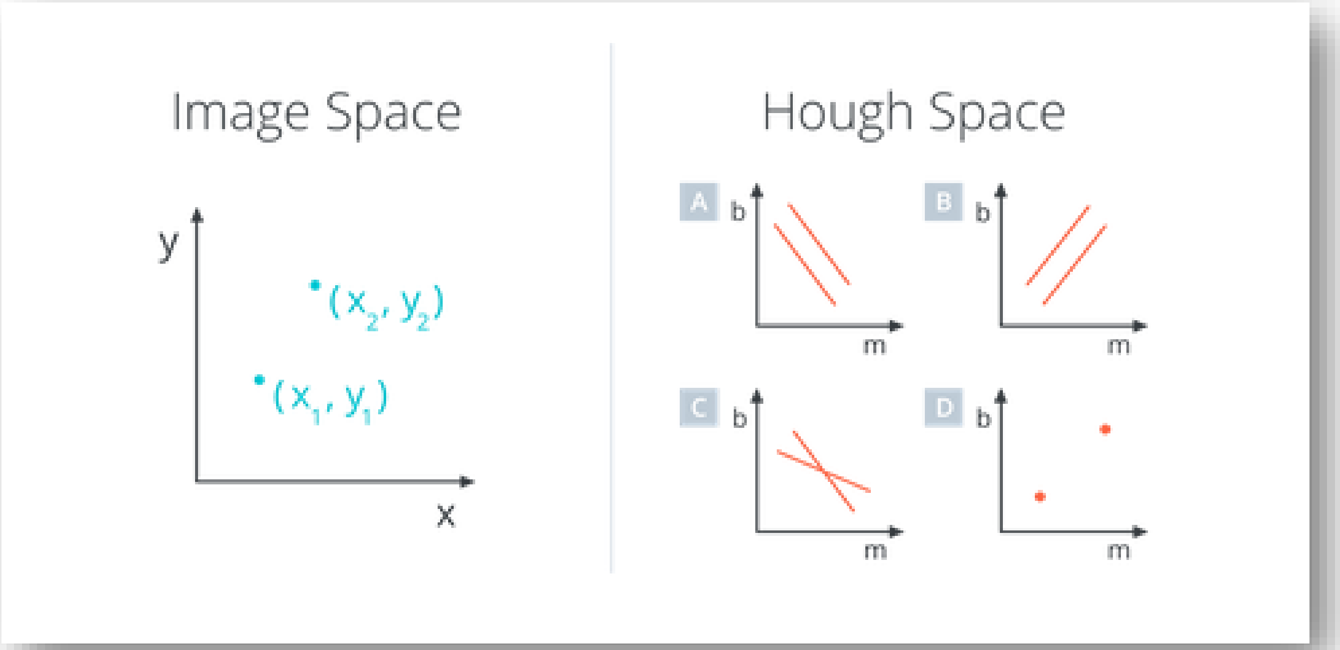


Question 2 of 5

What does a point in image space correspond to in Hough space?

- ● A
- ❍ B
- ❍ C
- ❍ D

Hough Transform



Question 3 of 5

What is the representation in Hough space of two points in image space?

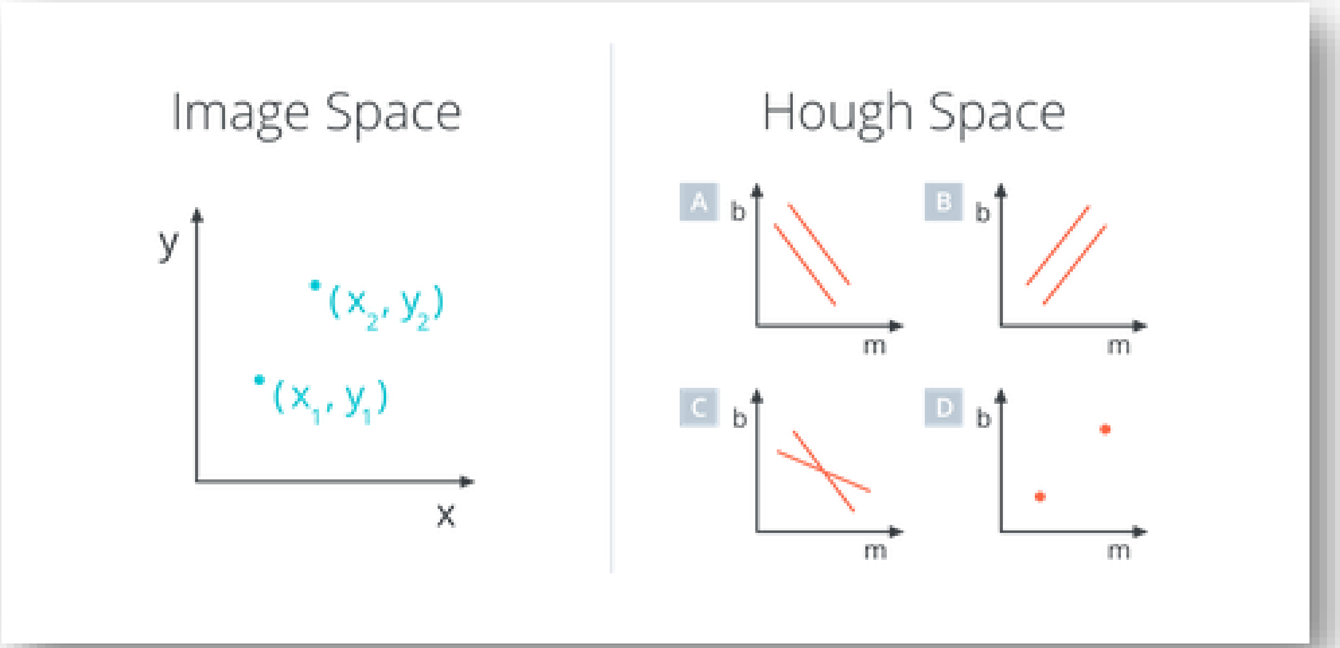
❍A

❍B

❍C

❍D

Hough Transform



Question 3 of 5

What is the representation in Hough space of two points in image space?

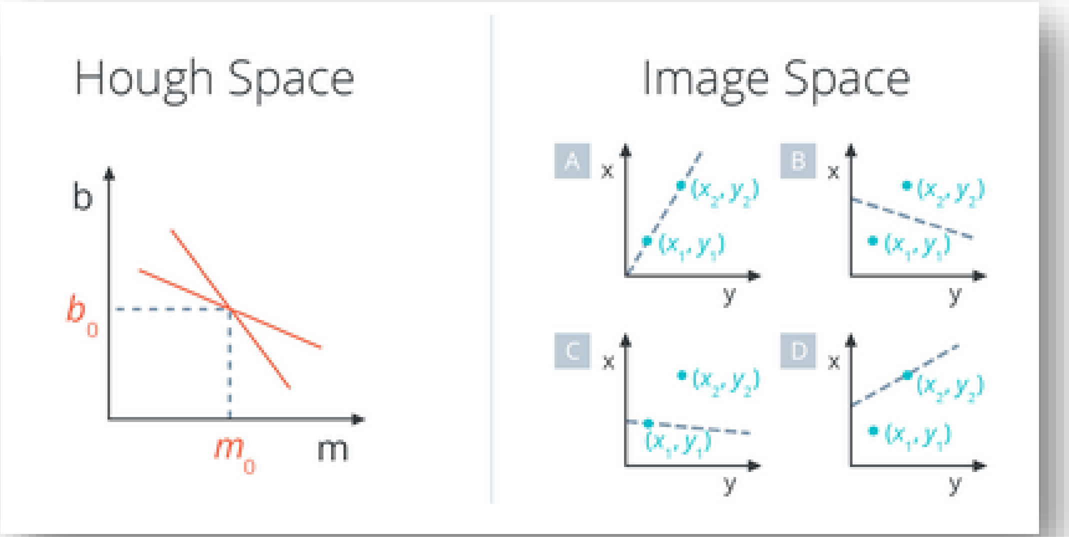
❍A

❍B

● C

❍D

Hough Transform



Question 4 of 5

What does the intersection point of the two lines in Hough space correspond to in image space?

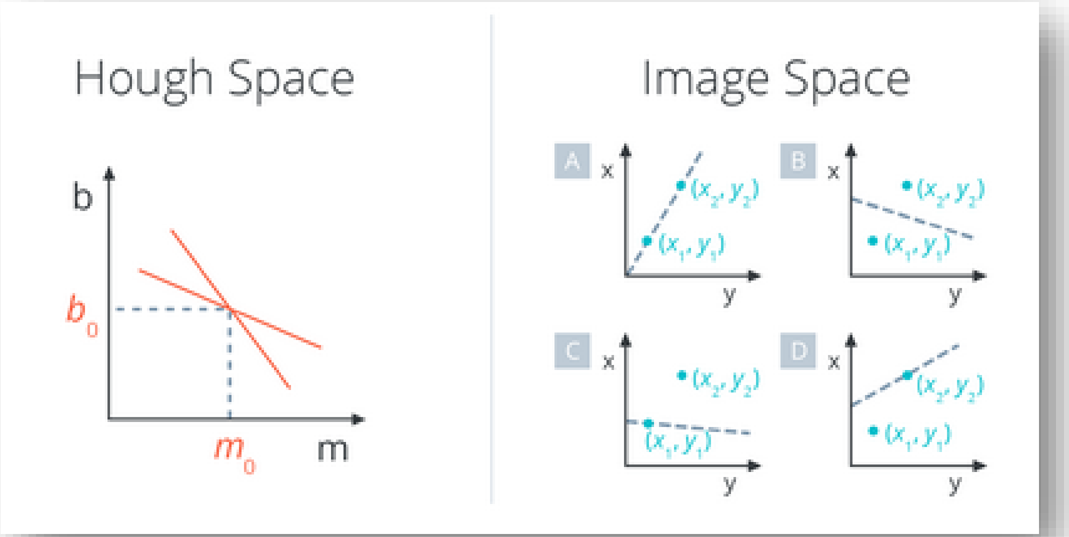
❍A) A line in image space that passes through both (x1,y1) and (x2,y2)

❍B) A line in image space that passes between (x1,y1) and (x2,y2)

❍C) A line in image space that passes through (x1,y1)

❍D) A line in image space that passes through only (x2,y2)

Hough Transform

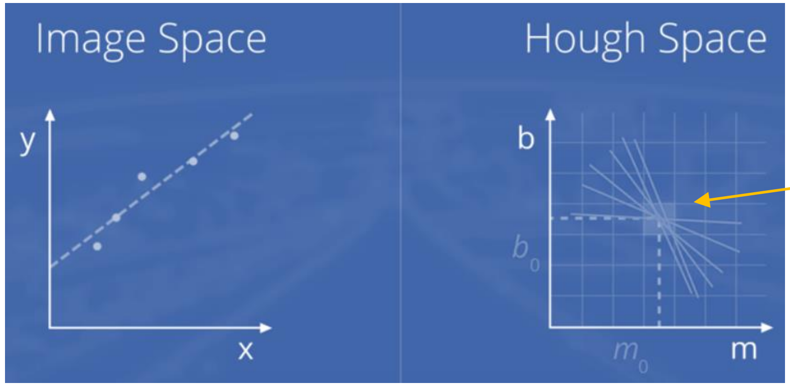


Question 4 of 5

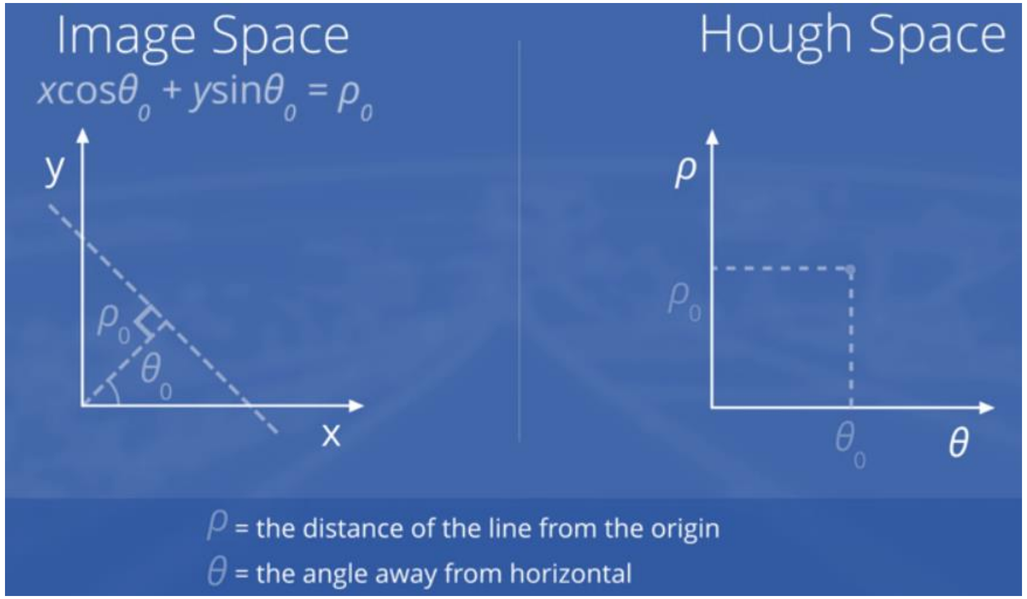
What does the intersection point of the two lines in Hough space correspond to in image space?

- ● A) A line in image space that passes through both (x1,y1) and (x2,y2)
- ❍ B) A line in image space that passes between (x1,y1) and (x2,y2)
- ❍ C) A line in image space that passes through (x1,y1)
- ❍ D) A line in image space that passes through only (x2,y2)

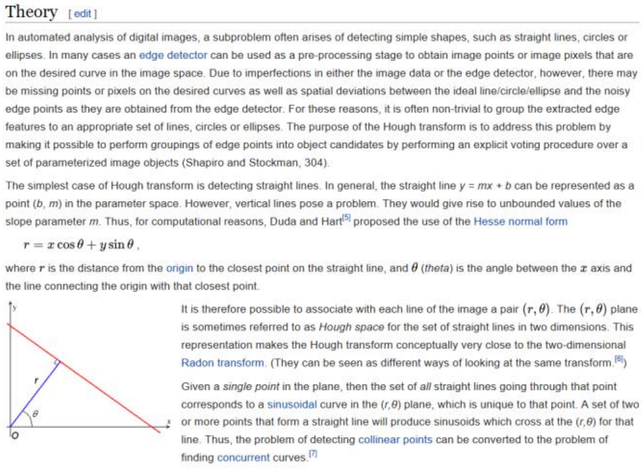
Hough Transform



Define intersecting lines as all lines passing through a given grid cell.



Problem: vertical lines have infinite slope in M-B representation. So we need a new parameterization.



Theory [edit]

In automated analysis of digital images, a subproblem often arises of detecting simple shapes, such as straight lines, circles or ellipses. In many cases an edge detector can be used as a pre-processing stage to obtain image points or image pixels that are on the desired curve in the image space. Due to imperfections in either the image data or the edge detector, however, there may be missing points or pixels on the desired curves as well as spatial deviations between the ideal line/circle/ellipse and the noisy edge points as they are obtained from the edge detector. For these reasons, it is often non-trivial to group the extracted edge features to an appropriate set of lines, circles or ellipses. The purpose of the Hough transform is to address this problem by making it possible to perform groupings of edge points into object candidates by performing an explicit voting procedure over a set of parameterized image objects (Shapiro and Stockman, 304).

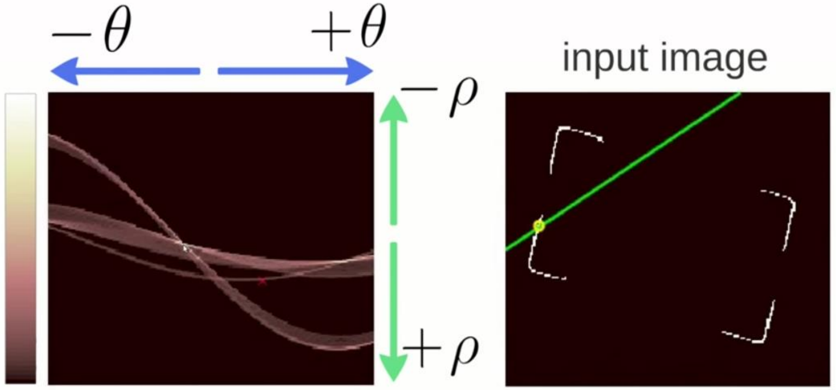
The simplest case of Hough transform is detecting straight lines. In general, the straight line $y = mx + b$ can be represented as a point (b, m) in the parameter space. However, vertical lines pose a problem. They would give rise to unbounded values of the slope parameter m. Thus, for computational reasons, Duda and Hart[5] proposed the use of the Hesse normal form

$$r = x\cos\theta + y\sin\theta\,,$$

where r is the distance from the origin to the closest point on the straight line, and θ (*theta*) is the angle between the x axis and the line connecting the origin with that closest point.

It is therefore possible to associate with each line of the image a pair (r, θ). The (r, θ) plane is sometimes referred to as *Hough space* for the set of straight lines in two dimensions. This representation makes the Hough transform conceptually very close to the two-dimensional Radon transform. (They can be seen as different ways of looking at the same transform.[6])

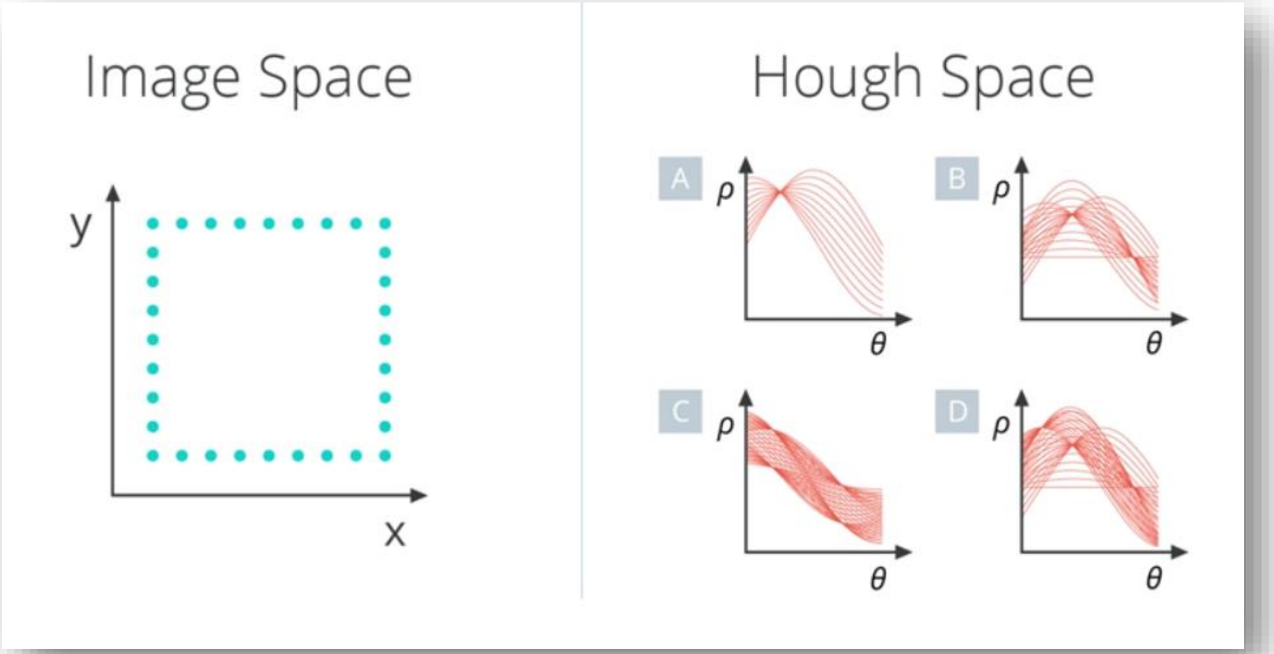
Given a *single point* in the plane, then the set of *all* straight lines going through that point corresponds to a sinusoidal curve in the (r,θ) plane, which is unique to that point. A set of two or more points that form a straight line will produce sinusoids which cross at the (r,θ) for that line. Thus, the problem of detecting collinear points can be converted to the problem of finding concurrent curves.[7]



Other reference: How Hough Transform works

reference: https://en.wikipedia.org/wiki/Hough_transform

Hough Transform

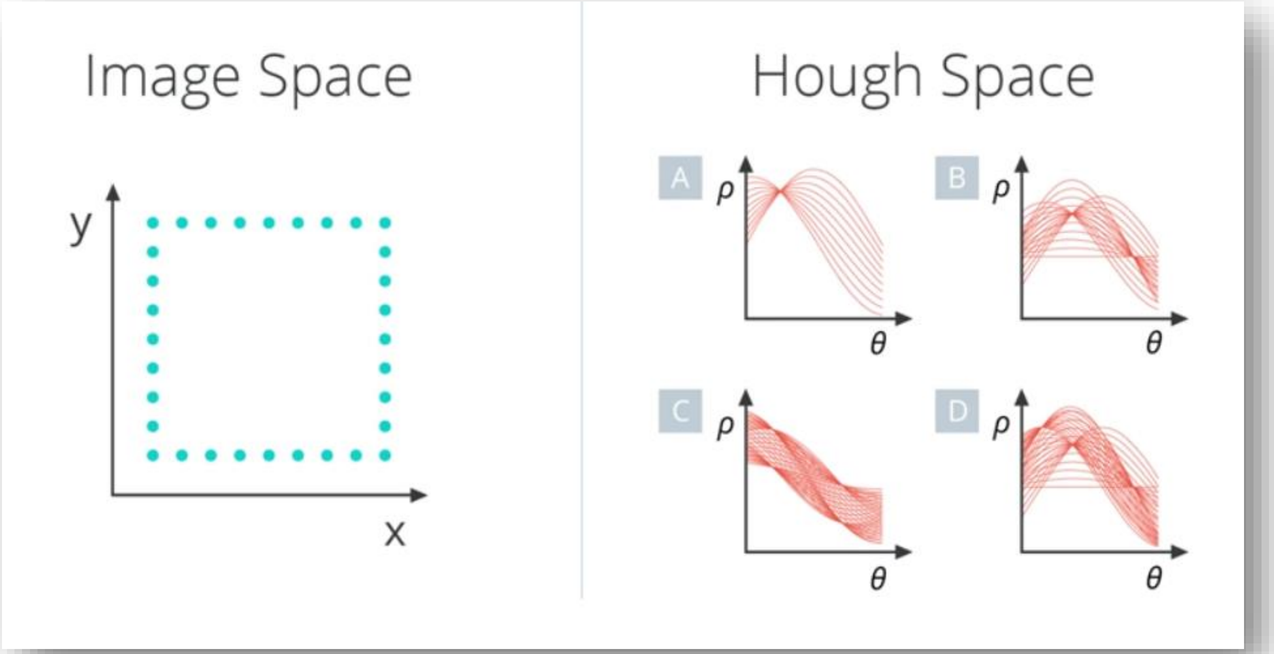


Question 5 of 5

What happens if we run a Hough Transform on an image of a square? What will the corresponding plot in Hough space look like?

❍A
❍B
❍C
❍D

Hough Transform



Question 5 of 5

What happens if we run a Hough Transform on an image of a square? What will the corresponding plot in Hough space look like?

- ❍ A
- ❍ B
- ● C
- ❍ D

Hough Transform to Find Lane Lines

OpenCV function: HoughLinesP

The output from HoughLinesP will be lines.

an array containing the endpoints (x1, y1, x2, y2)

The distance and [angular resolution]() of our grid in Hough space.

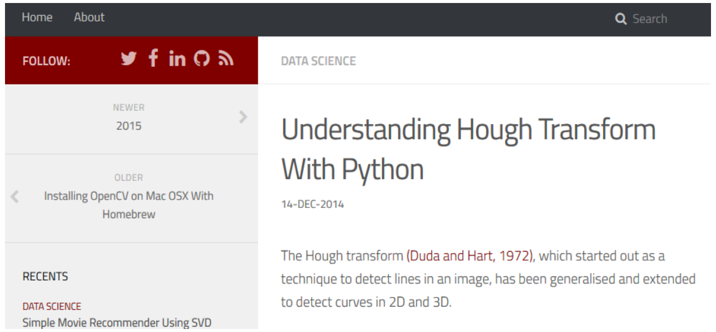
The empty np.array([]) is just a placeholder, no need to change it

The maximum distance connected into a single line

```
lines = cv2.HoughLinesP(edges, rho, theta, threshold, np.array([]), min_line_length, max_line_gap)
```

The threshold parameter specifies the minimum number of votes (intersections in a given grid cell) a candidate line needs to have to make it into the output.

The minimum length of a line (in pixels) that you will accept in (pixels) between segments that you will allow to be the output.



Home About Search

FOLLOW:

NEWER
2015

OLDER
Installing OpenCV on Mac OSX With Homebrew

RECENTS

DATA SCIENCE
Simple Movie Recommender Using SVD

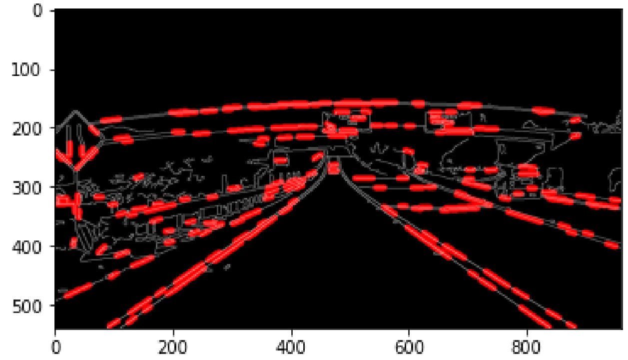
DATA SCIENCE

Understanding Hough Transform With Python

14-DEC-2014

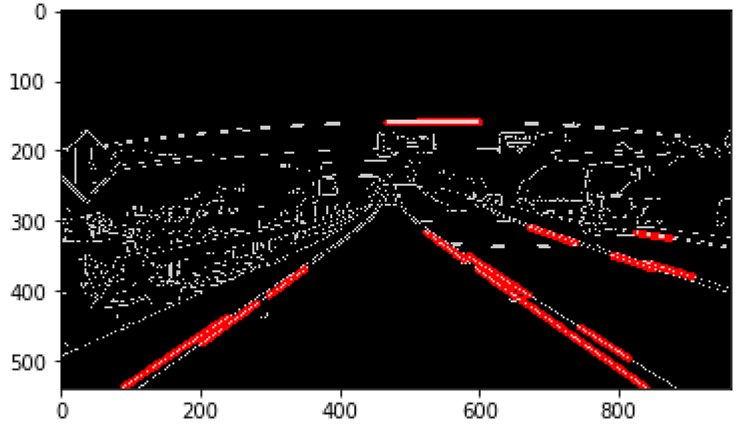
The Hough transform (Duda and Hart, 1972), which started out as a technique to detect lines in an image, has been generalised and extended to detect curves in 2D and 3D.

Code:
[05Hough_Transform_to_Find_Lane_Lines.txt]()



```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np import cv2 # Read in and grayscale the image image = mpimg.imread('exit-ramp.jpg')
gray = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY) # Define a kernel size and apply Gaussian smoothing
kernel_size = 5
blur_gray = cv2.GaussianBlur(gray,(kernel_size, kernel_size),0) # Define our parameters for Canny and apply
low_threshold = 50
high_threshold = 150
masked_edges = cv2.Canny(blur_gray, low_threshold, high_threshold)
```

```
# Define the Hough transform parameters # Make a blank the same size as our image to draw on
rho = 1 # distance resolution in pixels of the Hough grid
theta = np.pi/180 # angular resolution in radians of the Hough grid
threshold = 5 # minimum number of votes (intersections in Hough grid cell)
min_line_length = 50 #minimum number of pixels making up a line
max_line_gap = 1 # maximum gap in pixels between connectable line segments
line_image = np.copy(image)*0  #creating a blank to draw lines on
# Run Hough on edge detected image
lines = cv2.HoughLinesP(masked_edges, rho, theta, threshold, np.array([]), min_line_length,
max_line_gap)
# Iterate over the output "lines" and draw lines on the blank
for line in lines:
   for x1,y1,x2,y2 in line:
      cv2.line(line_image,(x1,y1),(x2,y2),(255,0,0),10)
# Create a "color" binary image to combine with line image
color_edges = np.dstack((masked_edges, masked_edges, masked_edges))
# Draw the lines on the edge image
combo = cv2.addWeighted(color_edges, 0.8, line_image, 1, 0)
plt.imshow(combo)
```

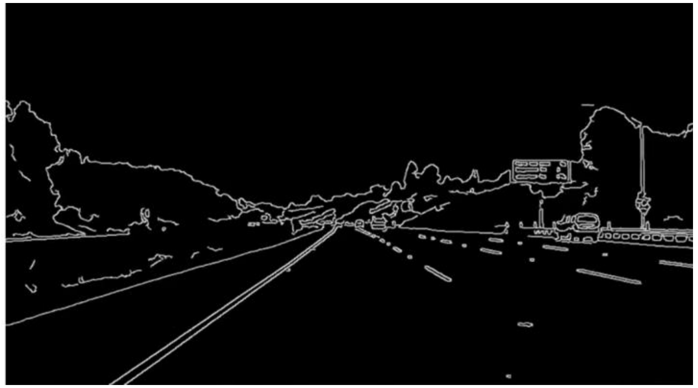


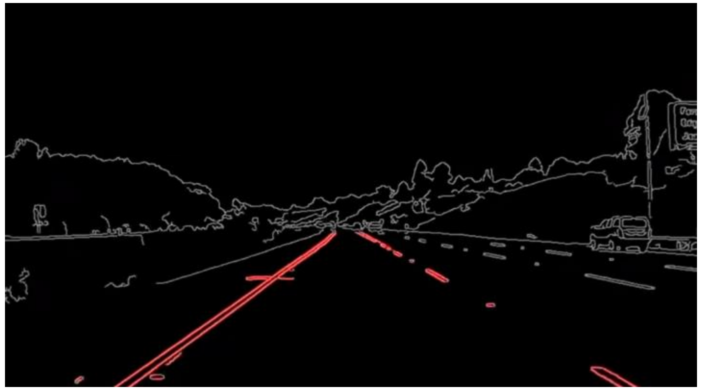
Hough Transform Quiz

The original image (left), edge detection and Hough transform (center), parameters optimized and region masked on the right.

Code: [05Quiz_Hough_Transform.txt]

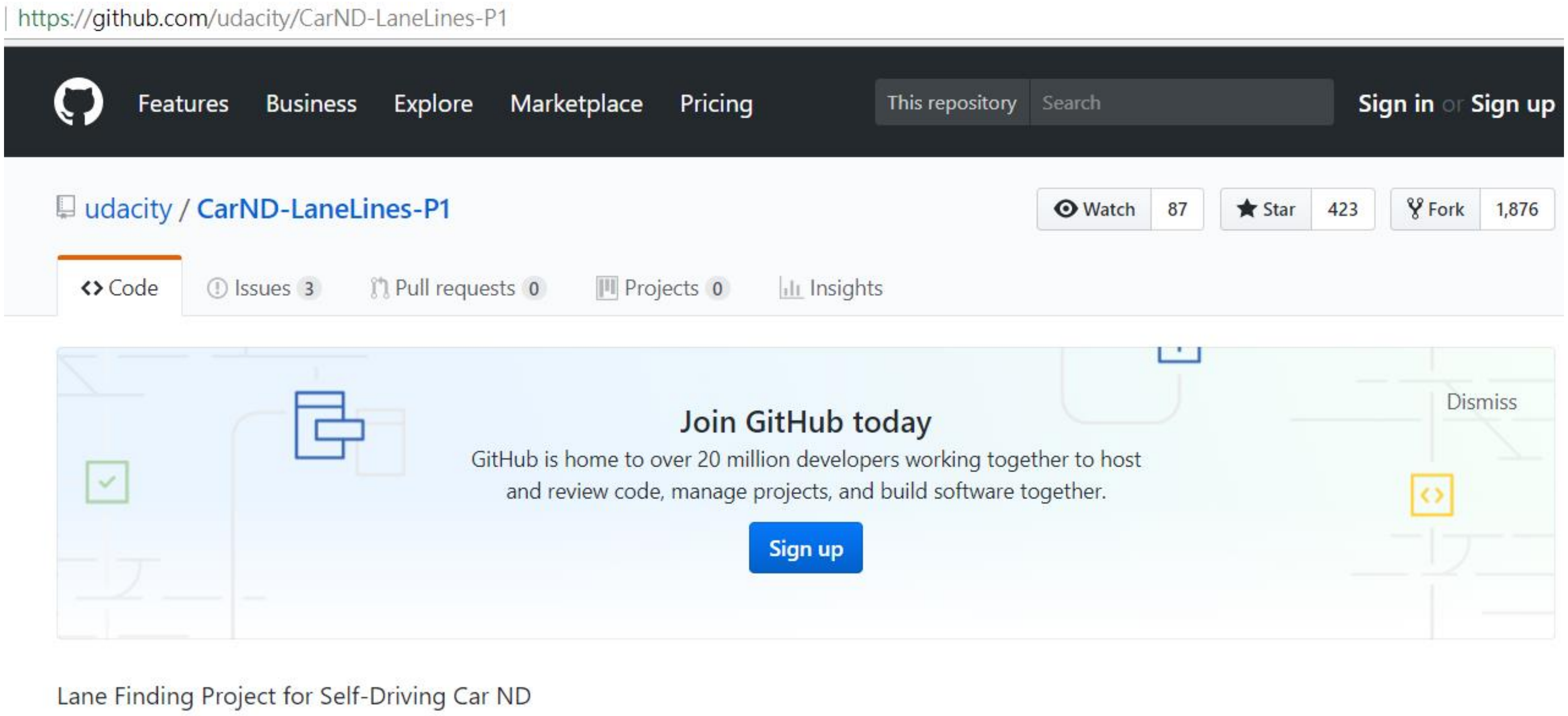
Project Intro: Finding Lane Lines in a Video Stream





Canny Edge Detection & Hough Transform

Project Intro: Finding Lane Lines in a Video Stream

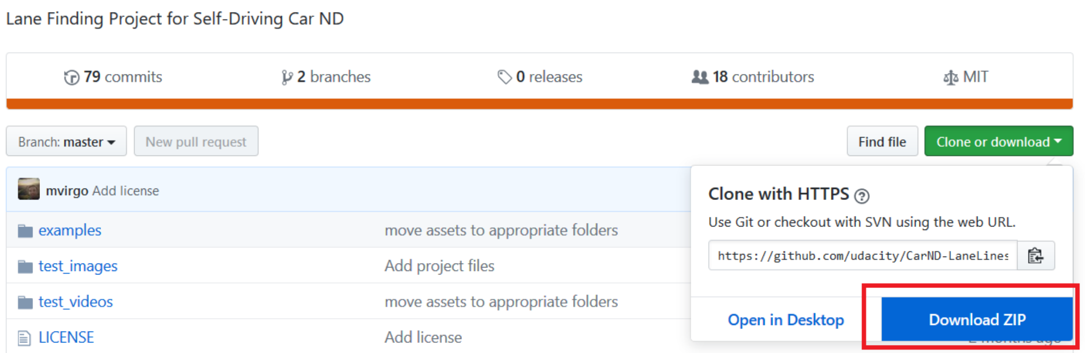


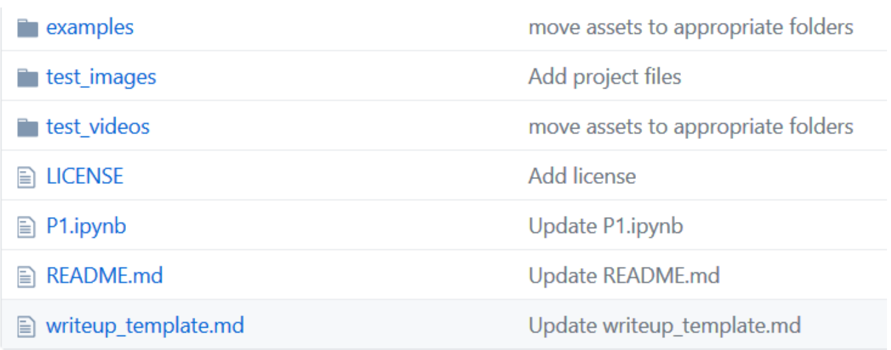
Reference: https://github.com/udacity/CarND-LaneLines-P1

Setup project

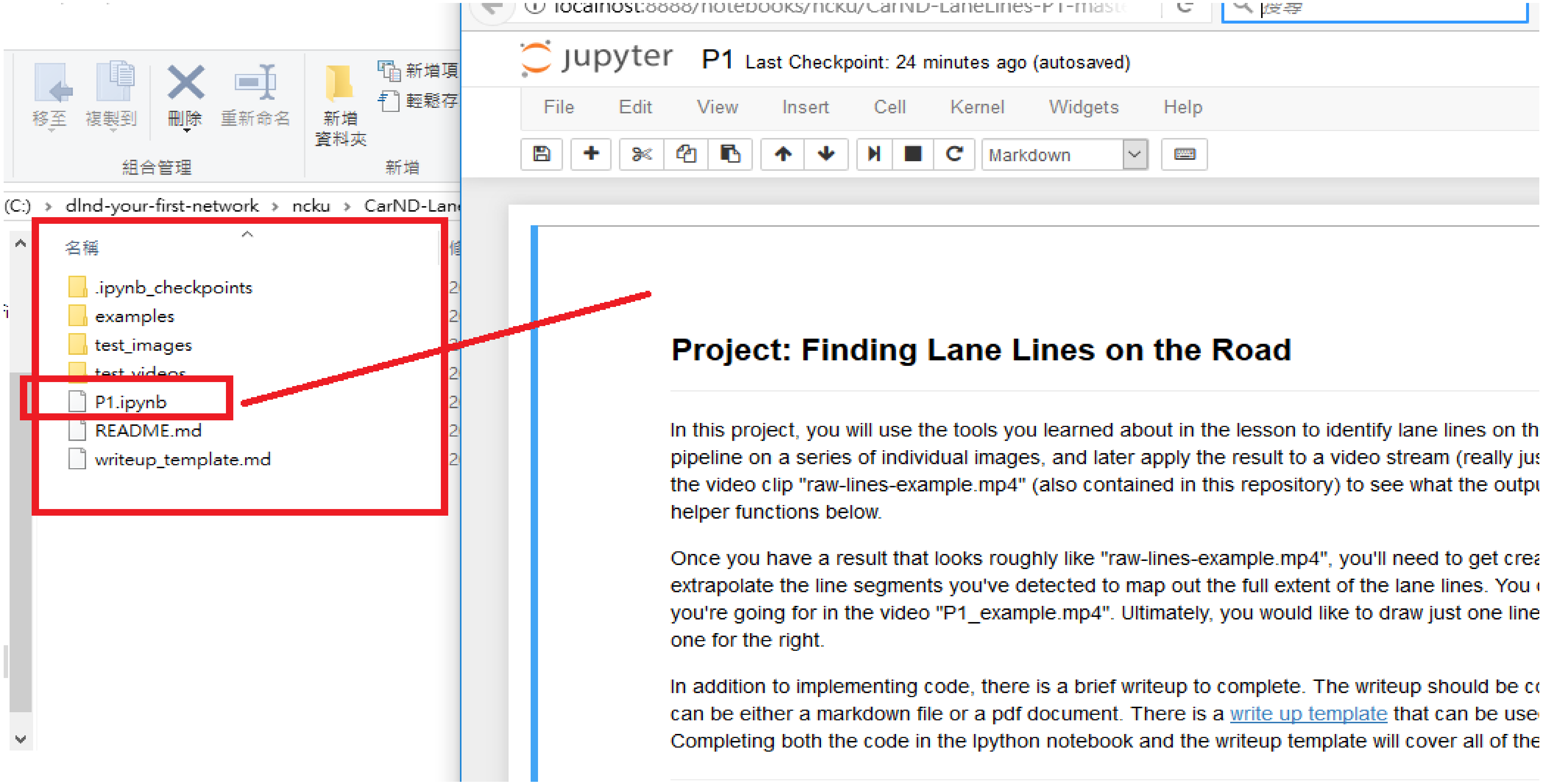
Instructions

1. Install miniconda or anaconda if you have not already.
2. Create an environment for "**Finding Lane Lines**"
   - Mac/Linux: conda create --name=xxxx python=2.7
   - Windows: conda create --name=xxxx python=3.7
3. Enter your conda environment
   - Mac/Linux: source activate xxxx
   - Windows: activate xxxx
4. conda install -c menpo opencv3 (conda install --channel https://conda.anaconda.org/menpo opencv3)
5. DownloadZip https://github.com/udacity/CarND-LaneLines-P1
6. extract zip file to a directory

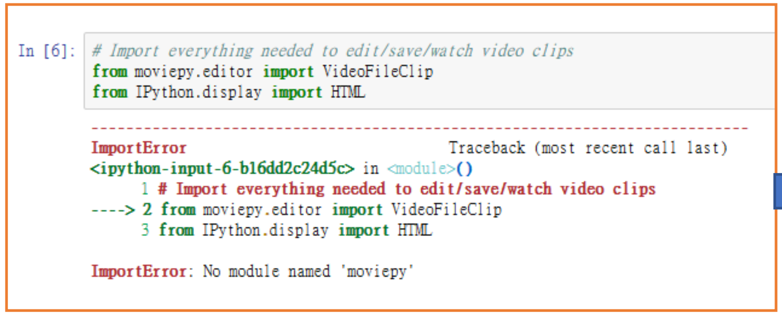


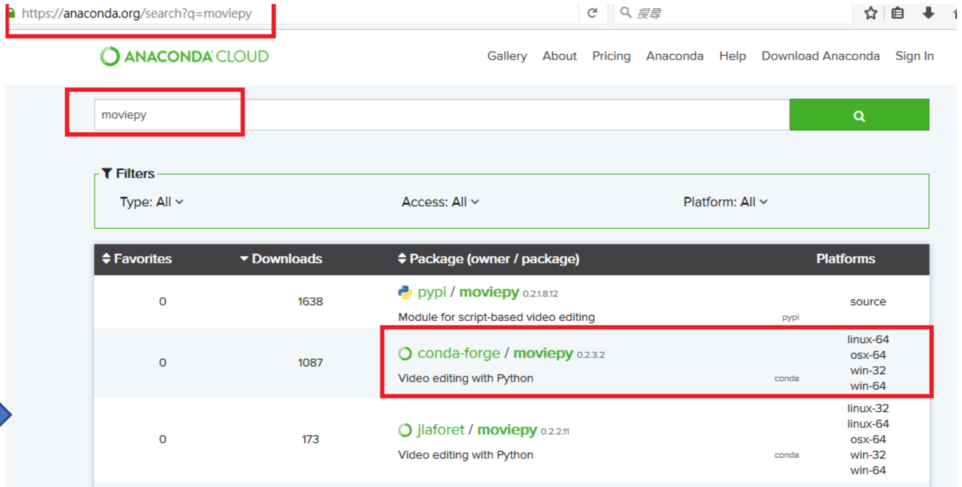


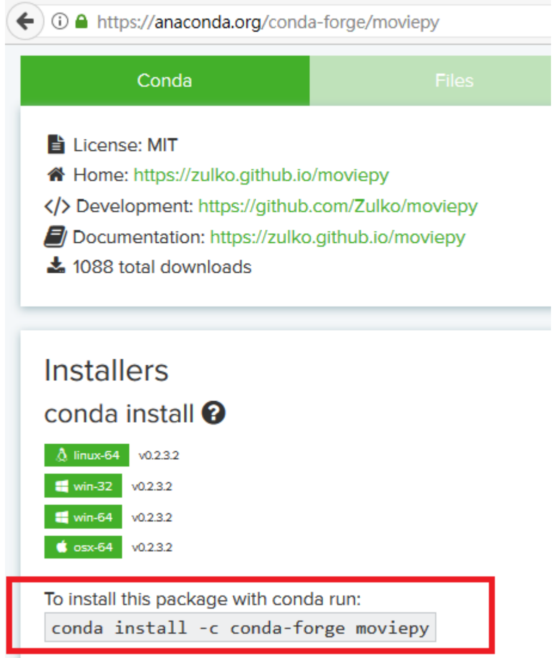
project

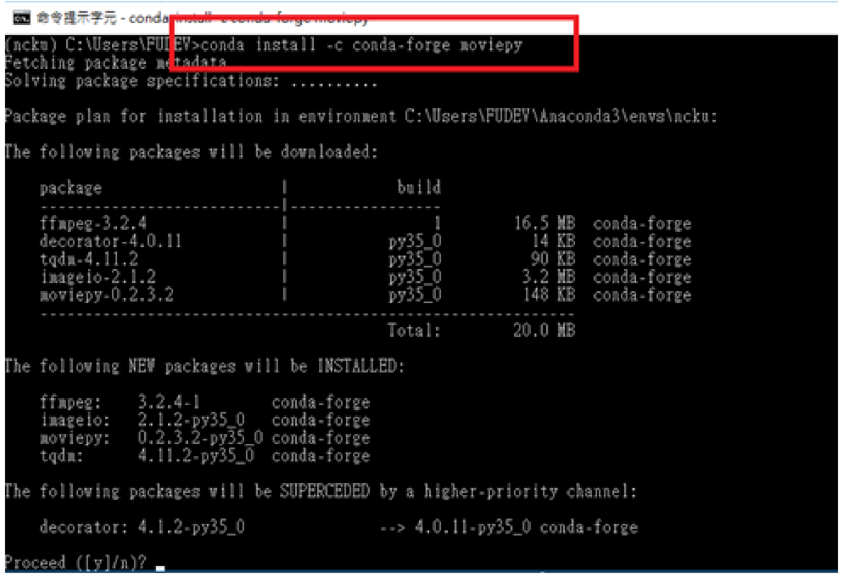

移至
複製到
刪除
重新命名
新增資料夾
新增項
輕鬆存
組合管理
新增
(C:) › dlnd-your-first-network › ncku › CarND-Lan
名稱
.ipynb_checkpoints
examples
test_images
test_videos
P1.ipynb
README.md
writeup_template.md
jupyter
P1 Last Checkpoint: 24 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Markdown
Project: Finding Lane Lines on the Road
In this project, you will use the tools you learned about in the lesson to identify lane lines on th
pipeline on a series of individual images, and later apply the result to a video stream (really jus
the video clip "raw-lines-example.mp4" (also contained in this repository) to see what the outpu
helper functions below.
Once you have a result that looks roughly like "raw-lines-example.mp4", you'll need to get crea
extrapolate the line segments you've detected to map out the full extent of the lane lines. You
you're going for in the video "P1_example.mp4". Ultimately, you would like to draw just one line
one for the right.
In addition to implementing code, there is a brief writeup to complete. The writeup should be co
can be either a markdown file or a pdf document. There is a write up template that can be use
Completing both the code in the Ipython notebook and the writeup template will cover all of the

Install moviepy



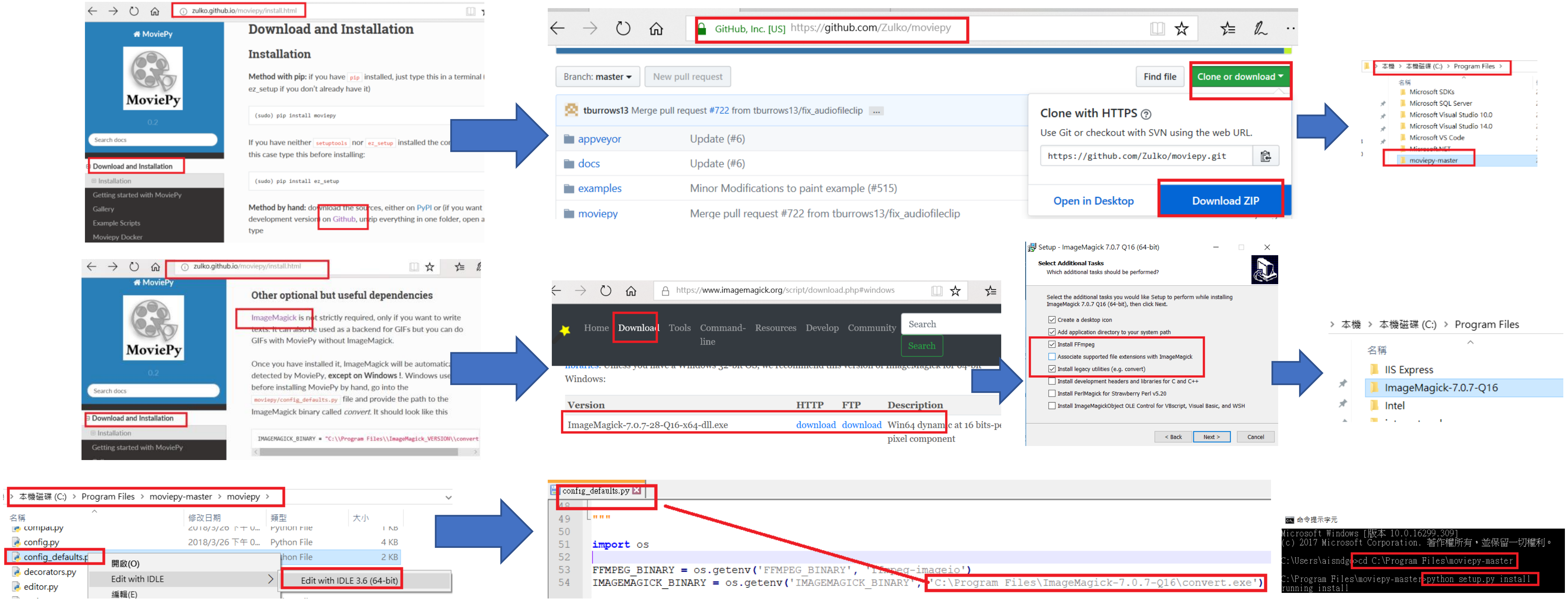






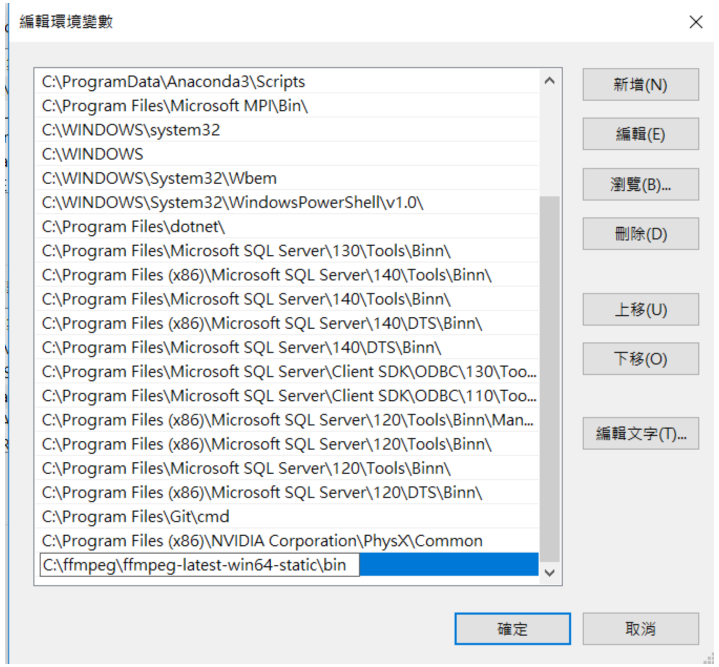
complete install Moviepy

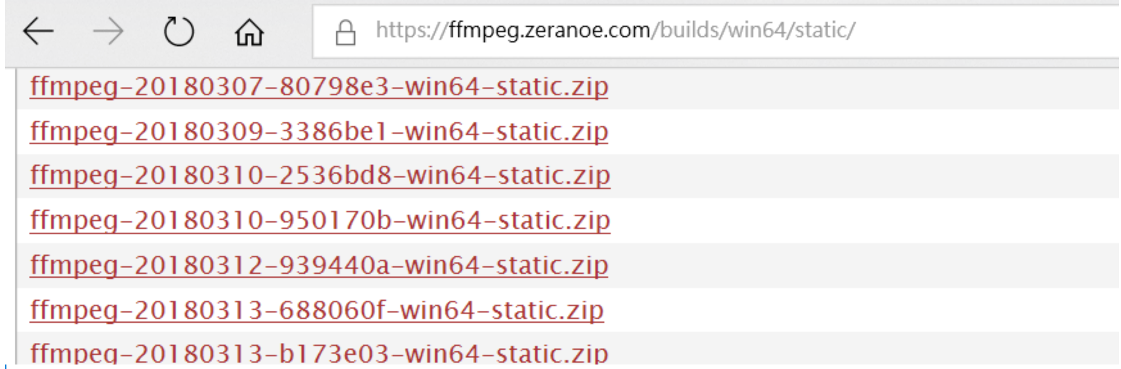
reference: http://zulko.github.io/moviepy/

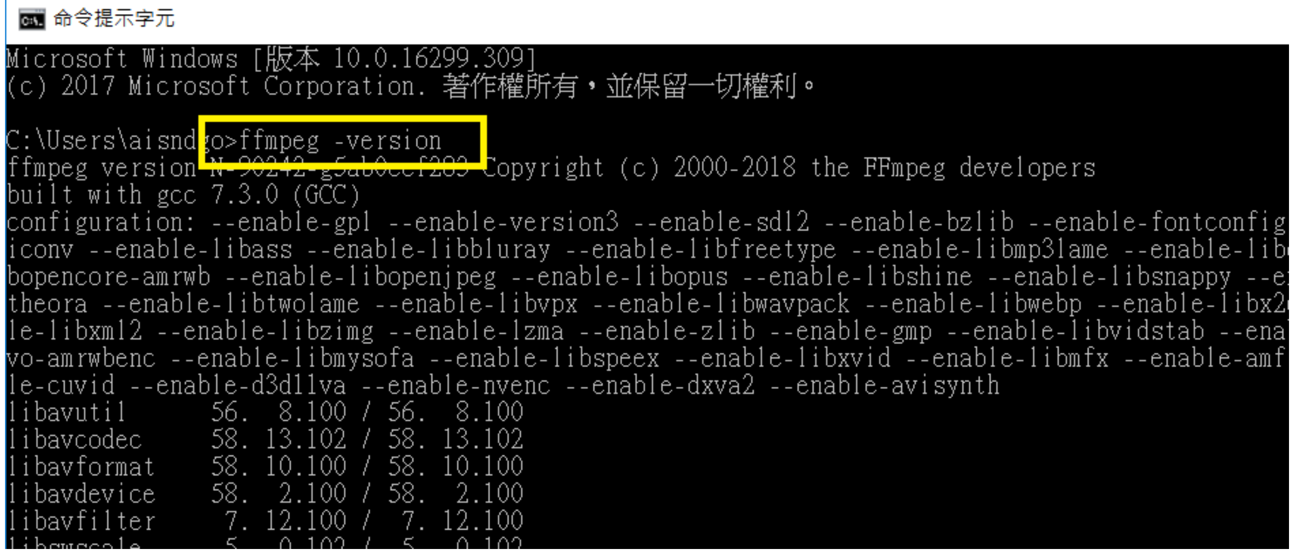


FFmpeg install

Download : https://www.ffmpeg.org/download.html







Project: Finding Lane Lines on the Road

- **The tools you have are color selection, region of interest selection, grayscaling, Gaussian smoothing, Canny Edge Detection and Hough Tranform line detection.**
- **Your goal is piece together a pipeline to detect the line segments in the image, then average/extrapolate them and draw them onto the image for display (as below). Once you have a working pipeline, try it out on the video stream below.**

Your output should look something like this (above) after detecting line segments using the helper functions below

Your goal is to connect/average/extrapolate line segments to get output like this

P1.ipynb Code

Import Packages

Read in an Image

Helper Functions

- grayscale
- canny
- gaussian_blur
- region_of_interest
- draw_lines
- hough_lines
- weighted_img

Test Images

make sure pipeline works well on these images before you try the videos

Build a Lane Finding Pipeline

1. define range of the color filter
2. get the color lane Pixels
3. get bounding box for region segmentation
4. get region filter grayscale
5. get edge image and add gausian blur to it with a kernel size of 3
6. combine the color filter pixels and the edge_filter_pixels
7. get hough image
8. overlay two images

Test on Videos

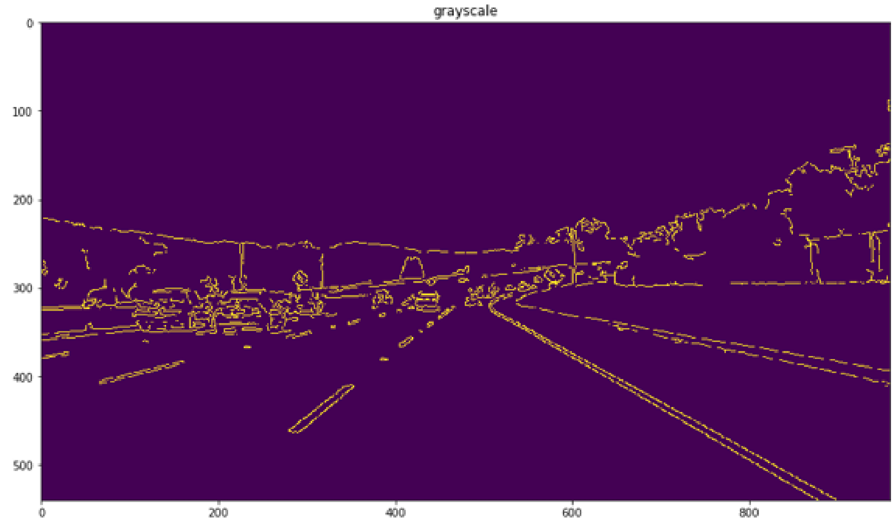
Test on Videos

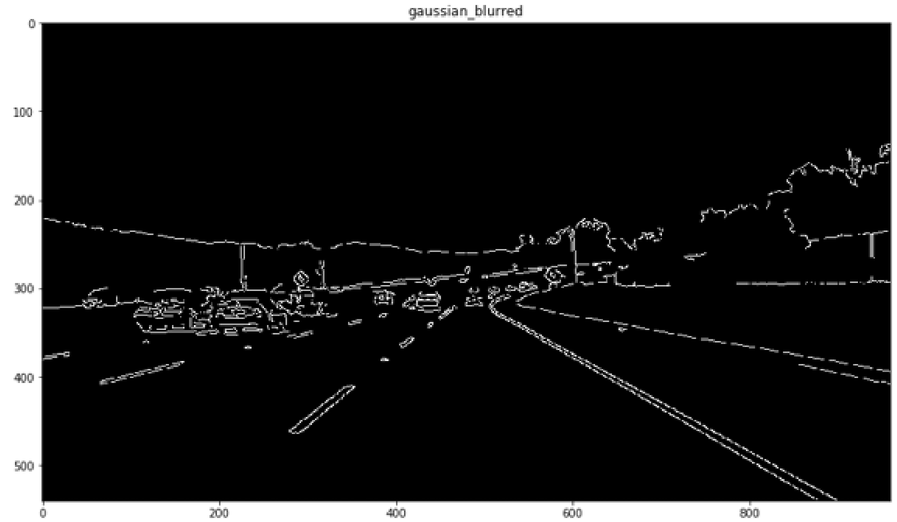
Optional Challenge

Test on Youtube Video

Canny edge detection and gaussian blur

```
view_from_windshield = mpimg.imread('edge.jpg')
working_copy = np.copy(view_from_windshield)
# first convert to grayscale
grayscale = cv2.cvtColor(working_copy, cv2.COLOR_RGB2GRAY)
#apply gaussian blur before canny edge detection
kernel_size=3
gaussian_blurred = cv2.GaussianBlur(grayscale,(kernel_size, kernel_size), 0)
# now apply canny edge detection threshold ratio 1:3
low_threshold = 60
high_threshold = 180
edges = cv2.Canny(gaussian_blurred, low_threshold, high_threshold)
edges2= cv2.Canny(grayscale, low_threshold, high_threshold)
f, (ax1, ax2) = plt.subplots(1,2, figsize=(30,20))
ax1.imshow(edges2)
ax1.set_title('grayscale')
ax2.imshow(edges,cmap='Greys_r')
ax2.set_title('gaussian_blurred')
plt.show()
```

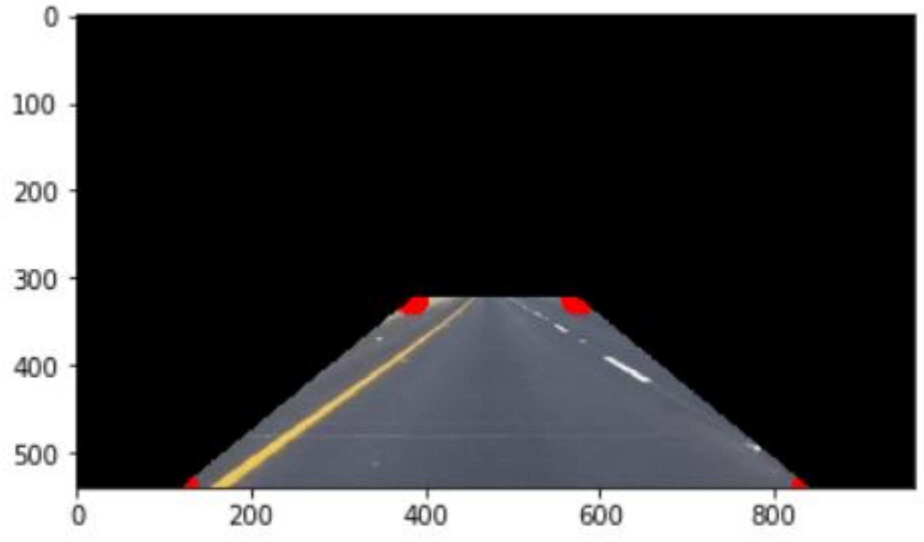




Region of interest(ROI)

```
def region_of_interest(img, vertices):
    """
    Applies an image mask.

    Only keeps the region of the image defined by the polygon
    formed from `vertices`. The rest of the image is set to black.
    """
    #defining a blank mask to start with
    mask = np.zeros_like(img)

    #defining a 3 channel or 1 channel color to fill the mask with depending on the input image
    if len(img.shape) > 2:
        channel_count = img.shape[2]  # i.e. 3 or 4 depending on your image
        ignore_mask_color = (255,) * channel_count
    else:
        ignore_mask_color = 255

    #filling pixels inside the polygon defined by vertices with the fill color
    cv2.fillPoly(mask, [vertices], ignore_mask_color)

    #returning the image only where mask pixels are nonzero
    masked_image = cv2.bitwise_and(img, mask)
    return masked_image
```

```
input_img_path='solidYellowCurve.jpg'
input_img_path = 'test_images/' + input_img_path
image = mpimg.imread(input_img_path)
ysize = image.shape[0]
xsize = image.shape[1]
bounding_box = get_bounding(xsize,ysize)
print(bounding_box)
cv2.circle(image,(120,540), 20, (255,0,0),-1)
cv2.circle(image,(384,324), 20, (255,0,0),-1)
cv2.circle(image,(576,324), 20, (255,0,0),-1)
cv2.circle(image,(840,540), 20, (255,0,0),-1)
plt.imshow(image)
plt.imshow(region_of_interest(image,bounding_box))
```

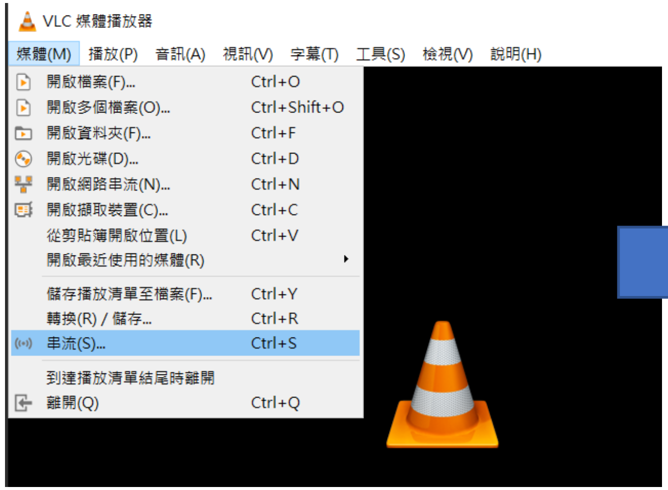


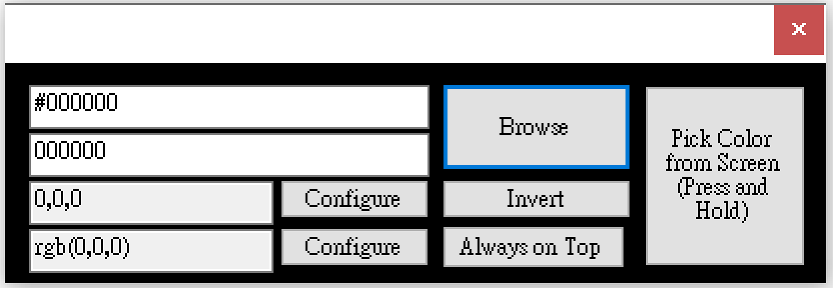
Test the function(detect Land Line)

https://www.youtube.com/watch?v=0crwED4yhBA&t=4708s =>(30 seconds)

YouTube MPEG-4
1280x720(16:9) 30p
640x480(4:3) 30p

h.264
h.265





Extract frames at different times form MP4

```
import os
from moviepy.editor import *

def extract_frames(movie, times, imgdir):
    clip = VideoFileClip(movie)
    for t in times:
        imgpath = os.path.join(imgdir, '{}.jpg'.format(t))
        clip.save_frame(imgpath, t)

movie = 'omg.mp4'
imgdir = 'frames'
times = 0.1, 0.63, 0.947, 1.2, 3.8, 6.7

extract_frames(movie, times, imgdir)
```

Advanced Lane Finding

- **Camera Calibration**
- **Apply a distortion correction to raw images**
- **Use color transforms, gradients, etc., to create a thresholded binary image**
- **Perspective transform**
- **Image Pipeline**
- **Detect lane pixels and fit to find the lane**
- **Test the function(detect Land & drawLine)**
- **Lane area drawing & Lane area drawing**
- **Determine the curvature of the lane and vehicle position with respect to center**
- **Test on Videos**