

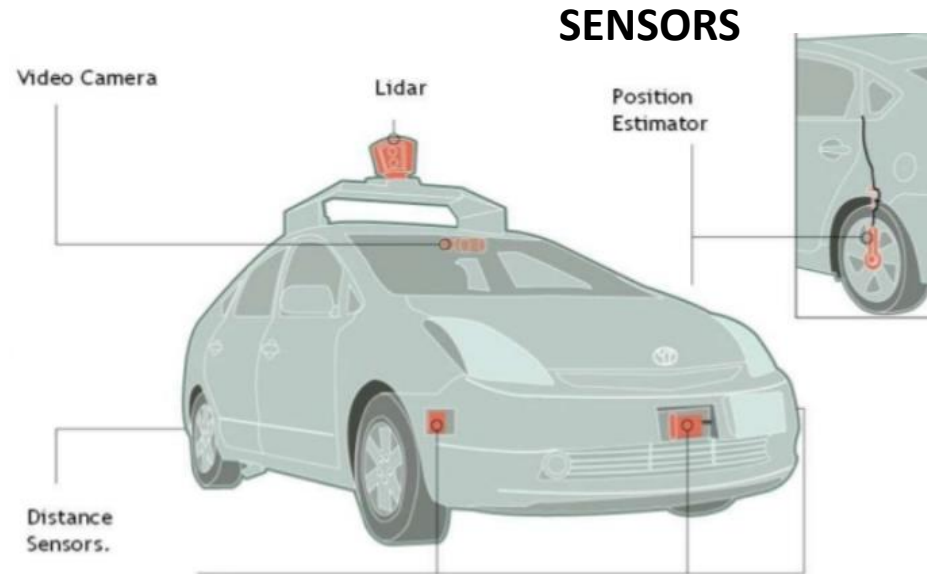
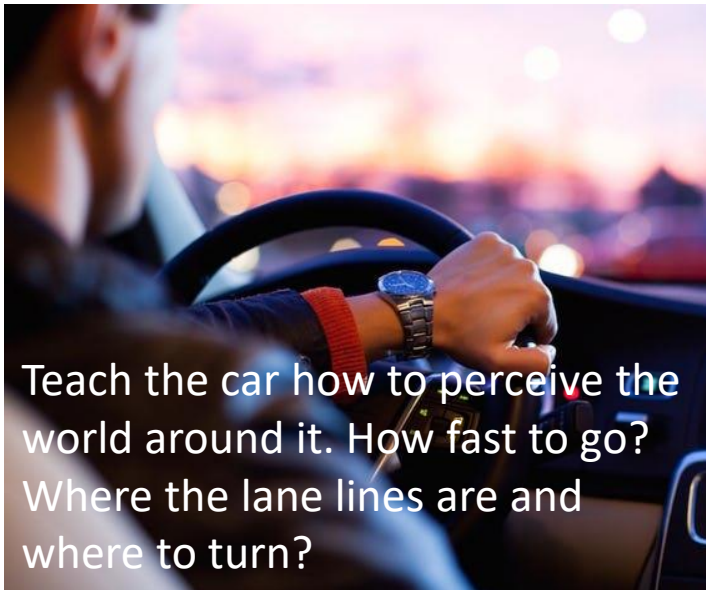
PERCEPTION FOR AUTONOMOUS CARS

2021/03/16

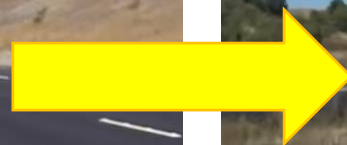
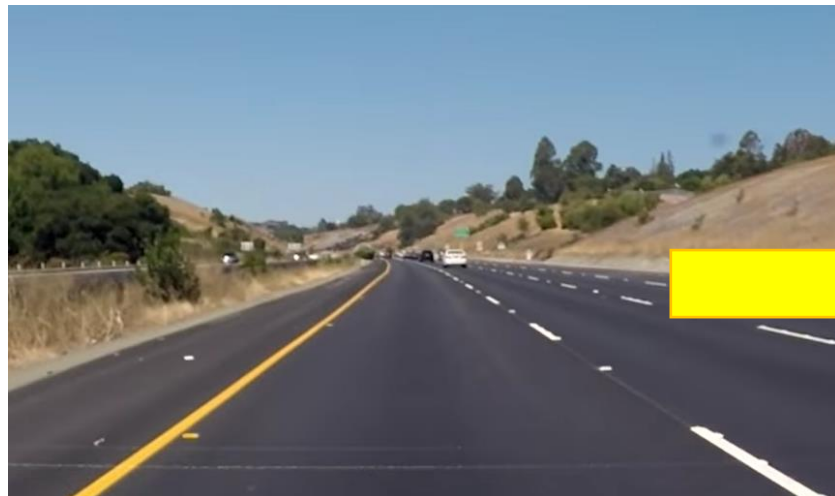
Finding Lane Lines

In this first lesson, you'll get taste of some basic computer vision techniques to find lane markings on the road. We will be diving much deeper into computer vision in later lessons.

Finding Lane Lines on the Road



I use my eyes to figure out how fast to go and where the lane lines are and where to turn. A car doesn't have eyes. But, in a self-driving car, we can use **cameras** and **other sensors** to achieve a similar function.



Your goal in this first module is to write code to **identify** and **track the position of the lane lines** in a series of images. And, in the project at the end of this first module, you're going to use image analysis techniques to do exactly that. So, let's start really high level. Here's a picture of a stretch of wide open highway. **What kinds of features** do you think would be helpful to figure out where the lane lines are in this image?

Finding Lane Lines on the Road



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

What kinds of features do you think would be helpful to figure out where the lane lines are in the image?

Identifying the lane lines using Color

How do we select the white pixels in an image?

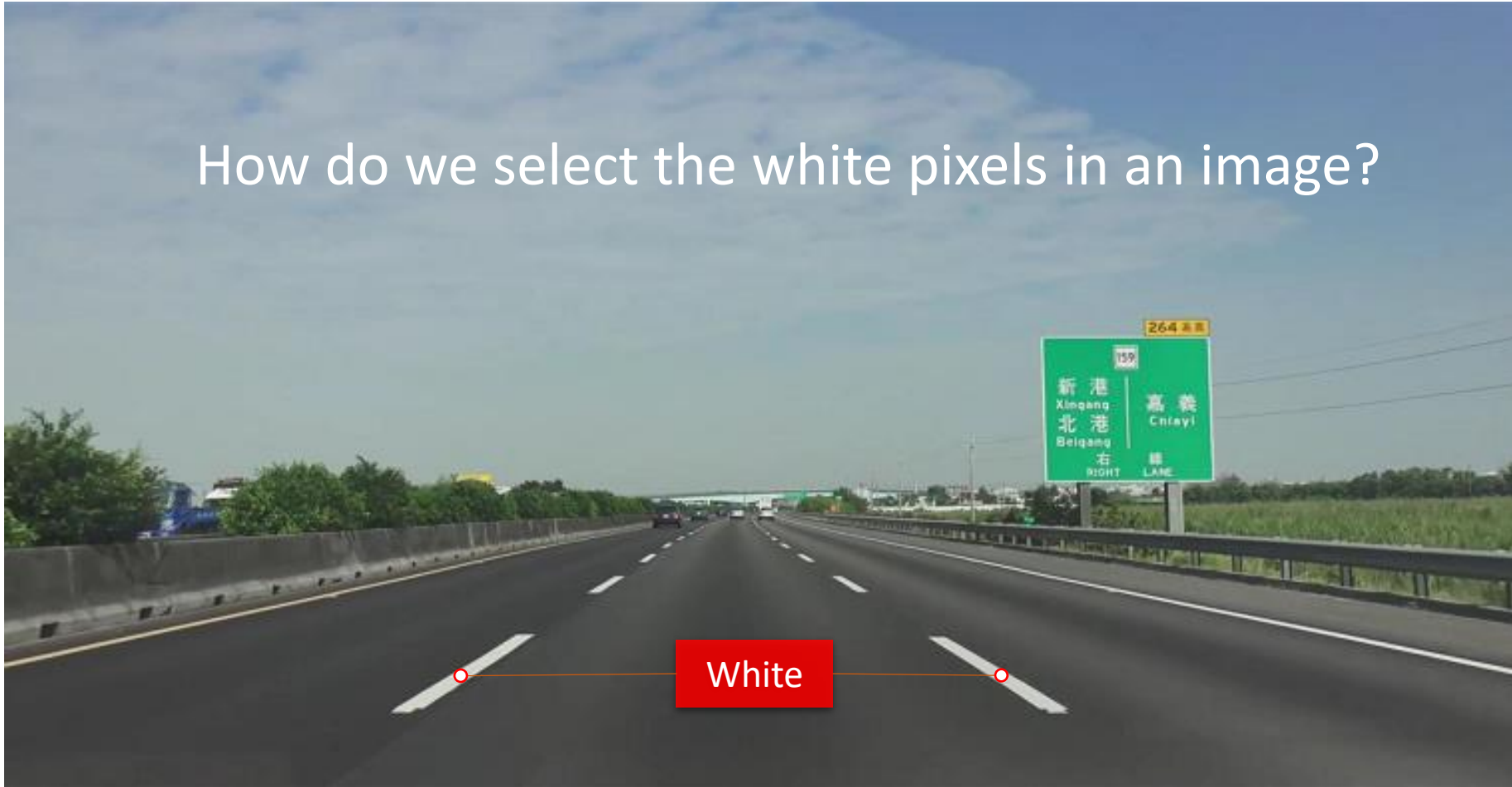
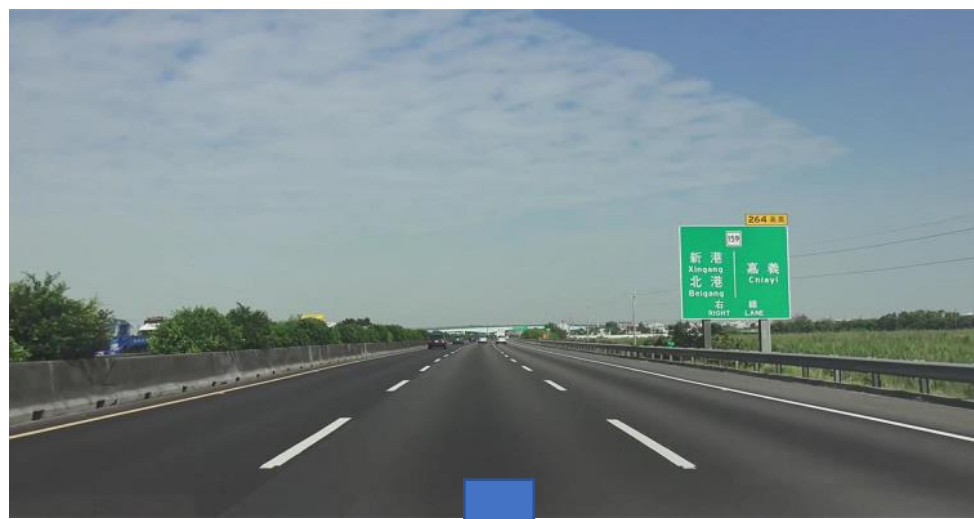
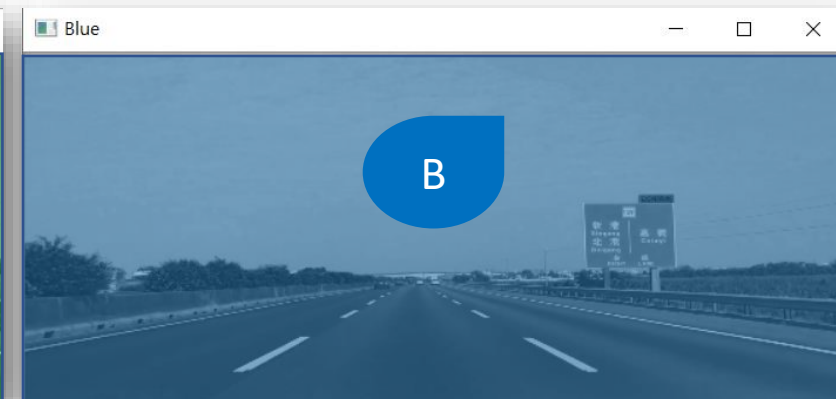
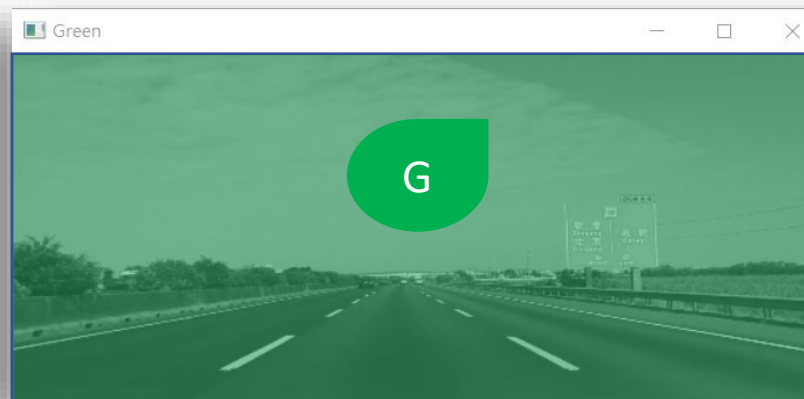
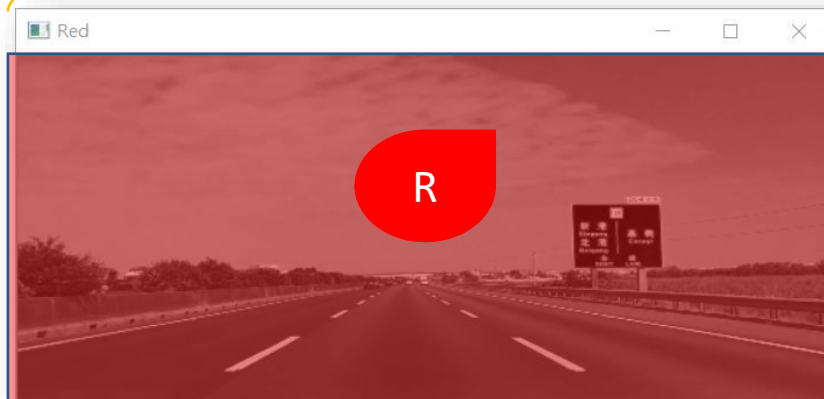


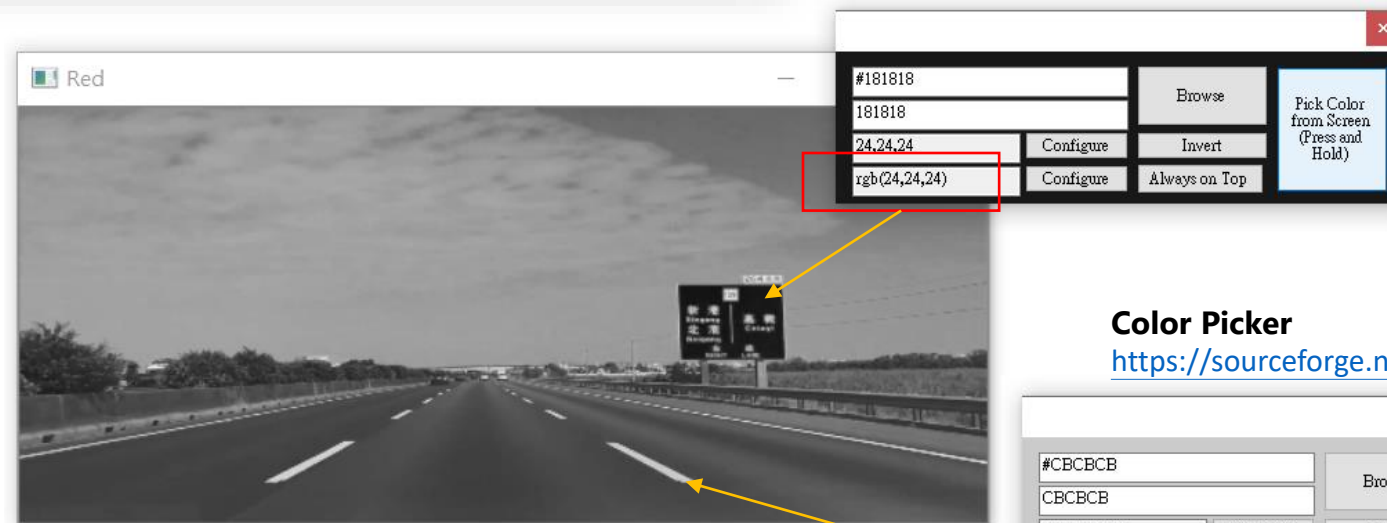
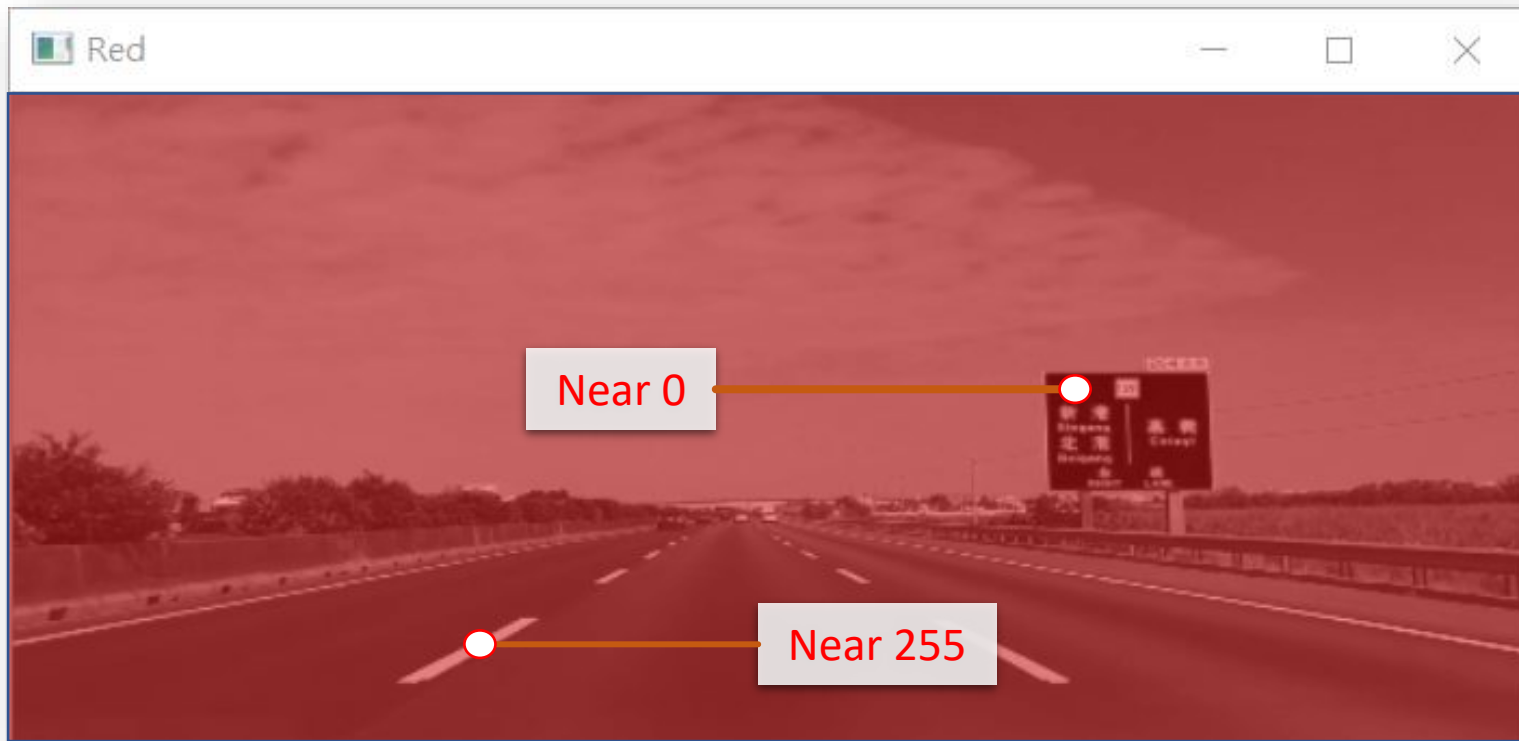
Image is actually
made up of a stack
of three images



```
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)
```

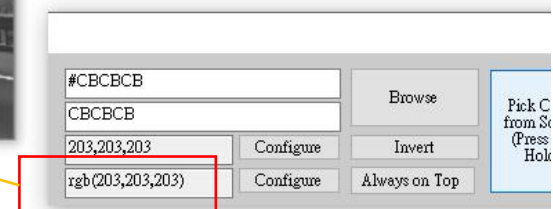




R G B 3 Color Chanel (0~255)

Color Picker

<https://sourceforge.net/projects/colorfinder/>





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: Color Selection Quiz



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

Color Selection Code Example

>Coding up a Color Selection

Coding up a Color Selection

Let's code up a simple color selection in Python.

No need to download or install anything, you can just follow along in the browser for now. ←

We'll be working with the same image you saw previously.



document: [Coding_up_a_Color_Selection.docx](#)

```
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
red_threshold = 0
green_threshold = 0
blue_threshold = 0
#####

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

```
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
```

`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)
```

Region Masking



- Now you've seen that with a simple color selection we have managed to eliminate almost everything in the image except the lane lines.
- At this point, however, it would still be tricky to extract the exact lines automatically, because we still have some other objects detected around the periphery that aren't lane lines.



Coding up a Region of Interest Mask

- In this case, I'll assume that the front facing camera that took the image is mounted in a fixed position on the car, such that the lane lines will always appear in the same general region of the image. Next, I'll take advantage of this by adding a criterion to only consider pixels for color selection in the region where we expect to find the lane lines.
- Check out the code below. The variables `left_bottom`, `right_bottom`, and `apex` represent the vertices of a triangular region that I would like to retain for my color selection, while masking everything else out. Here I'm using a triangular mask to illustrate the simplest case, but later you'll use a quadrilateral, and in principle, you could use any polygon.

Code: [02Region_Masking.txt](#)

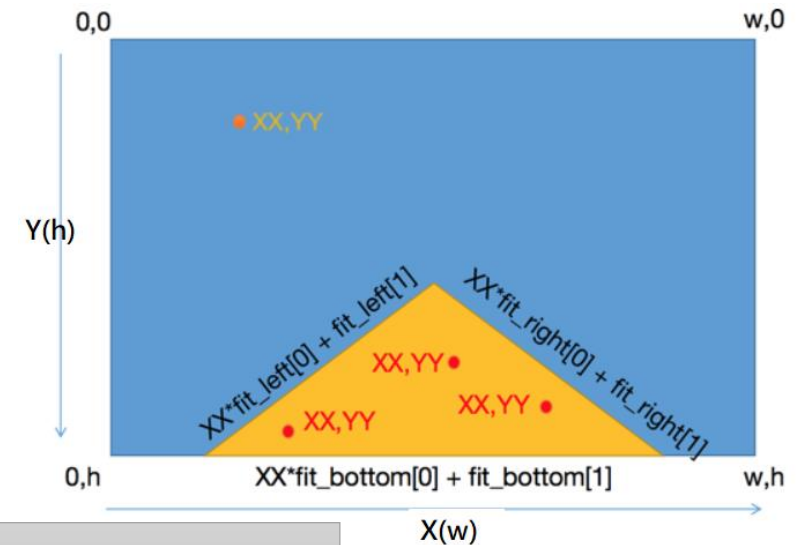
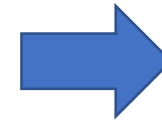
```
# Define a triangle region of interest.  
#Keep in mind the origin (x=0, y=0) is in the upper  
# left in image processing  
left_bottom = [0, 539]  
right_bottom = [900, 300]  
apex = [400, 0]
```

```
# 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 Region

Quiz: Color and Region Selection

In this next quiz, I've given you the values of `red_threshold`, `green_threshold`, and `blue_threshold` but now you need to modify `left_bottom`, `right_bottom`, and `apex` to represent the vertices of a triangle identifying the region of interest in the image. When you run the code in the quiz, your output result will be several images. Tweak the vertices until your output looks like the examples shown below.



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/api/pyplot_api.html#module-matplotlib.pyplot

Finding Lines of Any Color



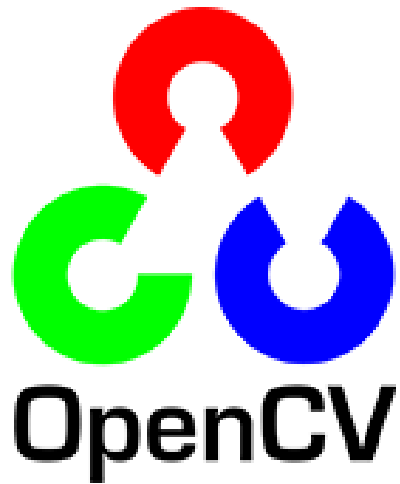
So you found the lane lines... simple right? Now you're ready to upload the algorithm to the car and drive autonomously right?? Well, not quite yet ;)

As it happens, 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.

So, what is computer vision?

What is Computer Vision?



The free Udacity course, [Introduction to 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.

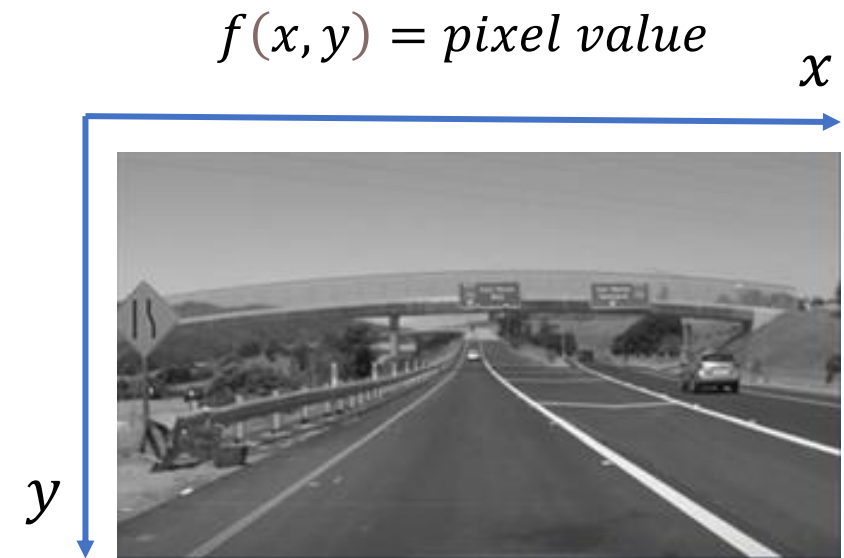


```
edges = cv2.Canny(gray, low_threshold, high_threshold)
```

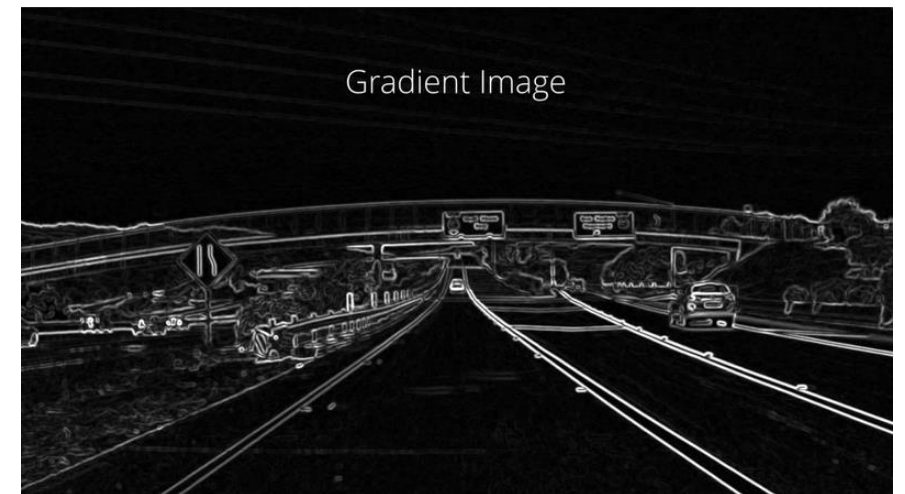
Canny and gradient



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



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



Canny Edge Detection

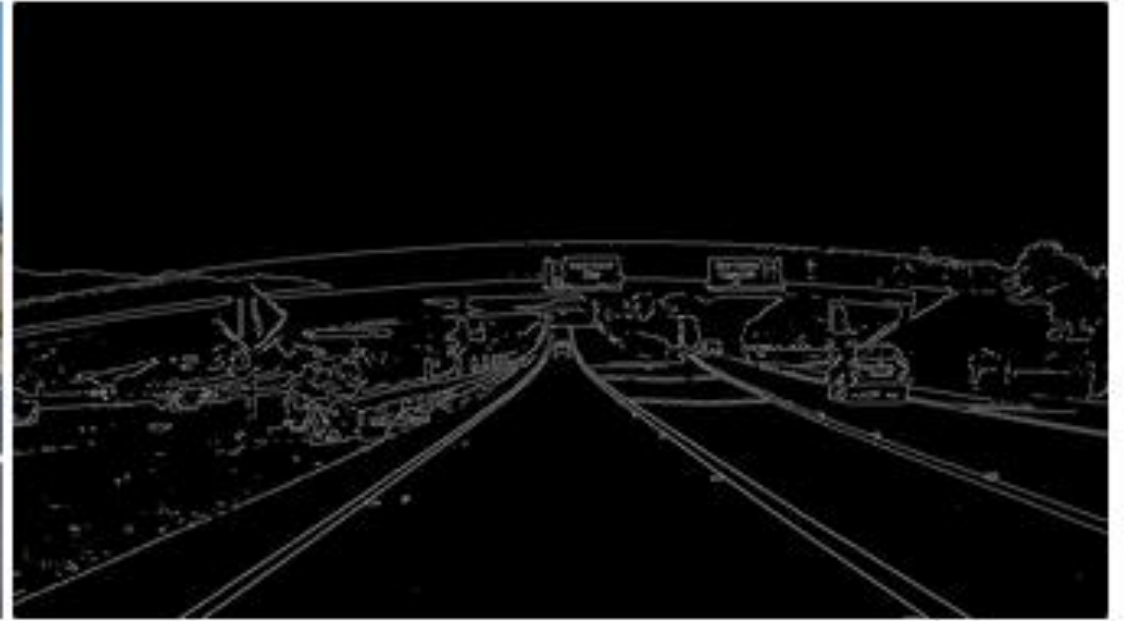
The red line in the plot right shows where I took a cross section through the image. The wiggles in the blue line indicate changes in intensity along that cross section through the image. Check all the boxes of the letters along this cross section, where you expect to find strong edges.

- ☐ A
- ☐ B
- ☐ C
- ☐ D
- ☐ E
- ☐ F



Canny Edges

Quiz: Canny Edge Detection Quiz



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')
```



Let's try our Canny edge detector on this image. This is where OpenCV gets useful. First, we'll have a look at the parameters for the OpenCV Canny function. You will call it like this:

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

補充:[補充CannyDetect.docx](#)

Code: [04Canny_to_Detect_Lane_Lines.txt](#)

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

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

```
# 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)
```

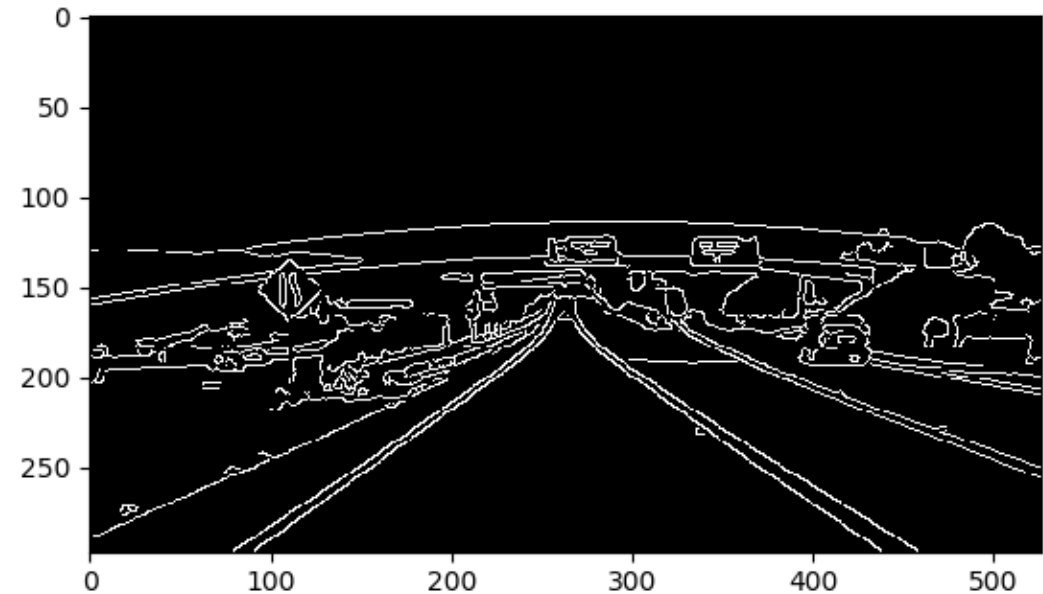
```
# Define parameters for Canny and run it
# NOTE: if you try running this code you might want to change
these!
low_threshold = 1
high_threshold = 10
edges = cv2.Canny(blur_gray, low_threshold, high_threshold)

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

Answer:

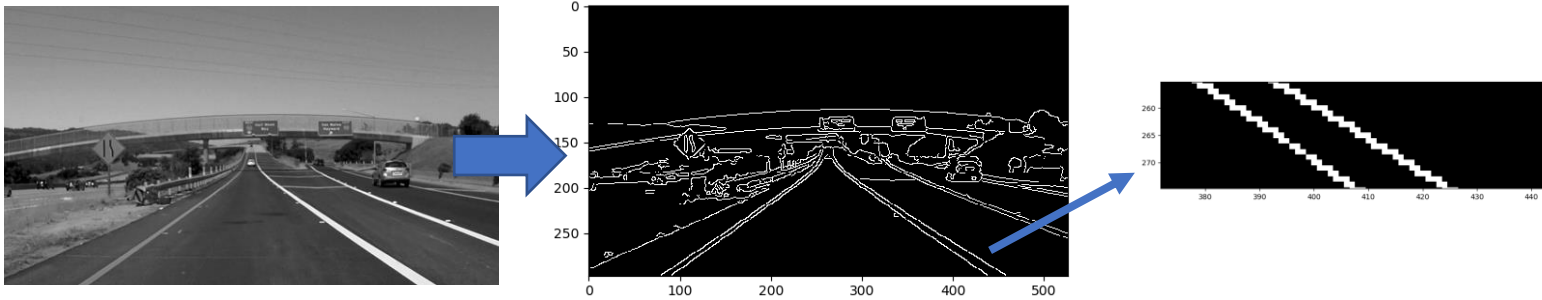
Here's how I did it...

I chose a kernel Size of 5 for Gaussian smoothing, a low Threshold of 50 and a high Threshold of 150. These selections nicely extract the lane lines, while minimizing the edges detected in the rest of the image, producing the result shown below.

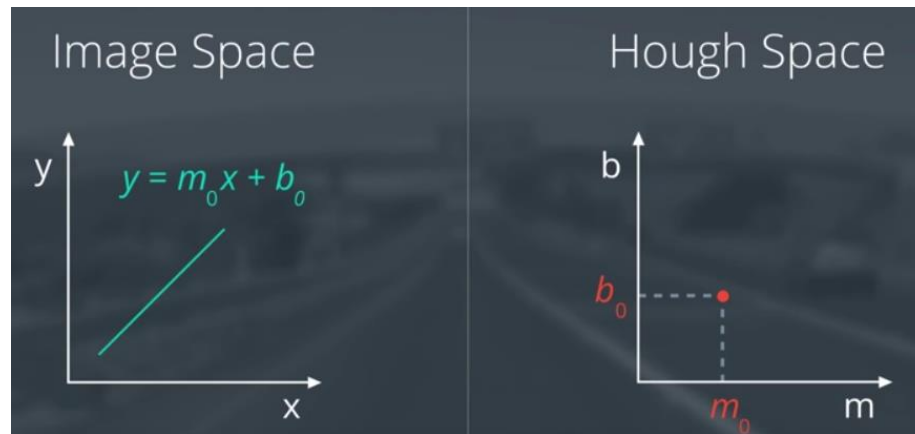


Hough Transform

Using the Hough Transform to Find Lines from Canny Edges



Using edge detection turned the greyscaled image into an image full of dots, but only **the dots that represent edges in the original image**. (Let's connect the dots.)

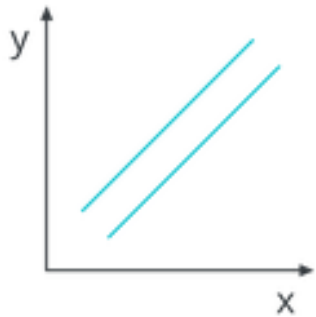


- In image space, a line is plotted as x vs. y , but in 1962, Paul Hough devised a method for representing lines in parameter space, which we will call “Hough space” in his honor.
- In Hough space, I can represent my “ x vs. y ” line as a point in “ m vs. b ” instead. The Hough Transform is just the conversion from image space to Hough space. So, the characterization of a line in image space will be a single point at the position (m, b) in Hough space.

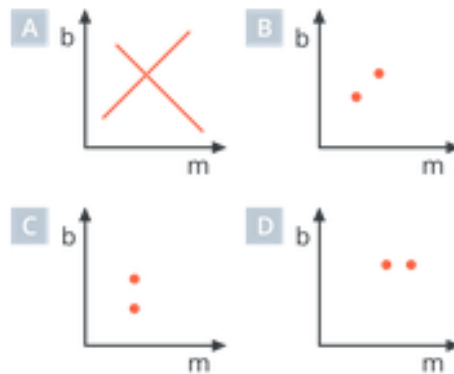
Hough Transform

So now I'd like to check your intuition... if a **line** in image space corresponds to a **point** in Hough space, what would **two parallel lines** in image space correspond to in Hough space?

Image Space



Hough Space



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

A line in image space corresponds to a point in Hough space. What does a point in image space correspond to in Hough space?

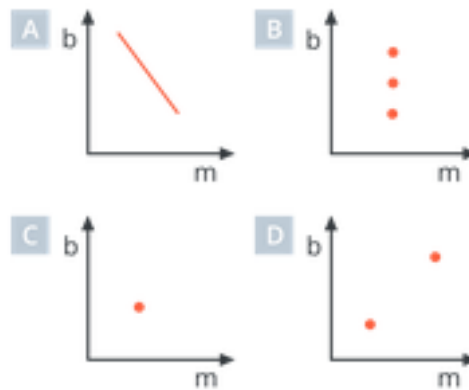
A single point in image space has many possible lines that pass through it, but not just any lines, only those with particular combinations of the m and b parameters. Rearranging the equation of a line, we find that a single point (x, y) corresponds to the line $b = y - xm$.

So what is the representation of a **point** in image space in Hough space?

Image Space



Hough Space



Question 2 of 5

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

- ☐ A
- ☐ B
- ☐ C
- ☐ D

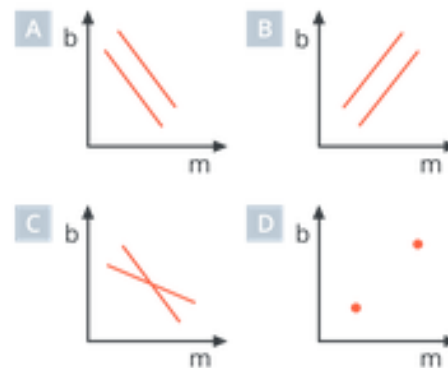
Hough Transform

What if you have **2 points** in image space. What would that look like in Hough space?

Image Space



Hough Space



Question 3 of 5

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

- ☐ A
- ☐ B
- ☐ C
- ☐ D

Hough Transform

Alright, now we have two intersecting lines in Hough Space. How would you represent their **intersection** at the point (m_0, b_0) in image space?

Hough Space

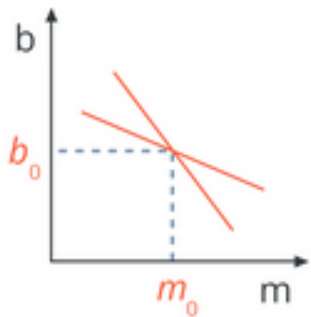
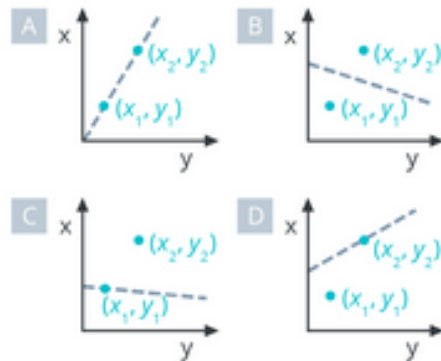


Image Space

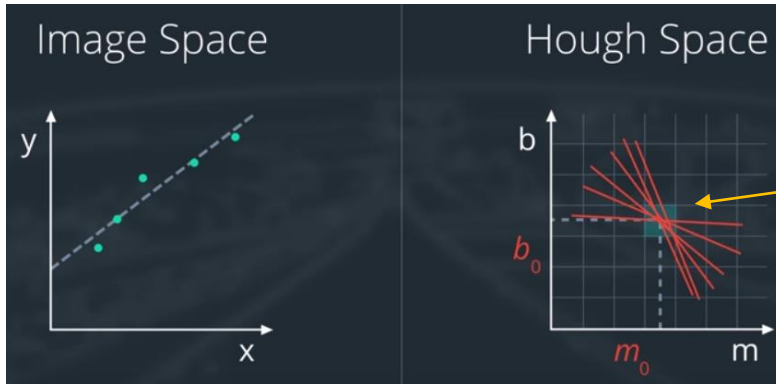


Question 4 of 5

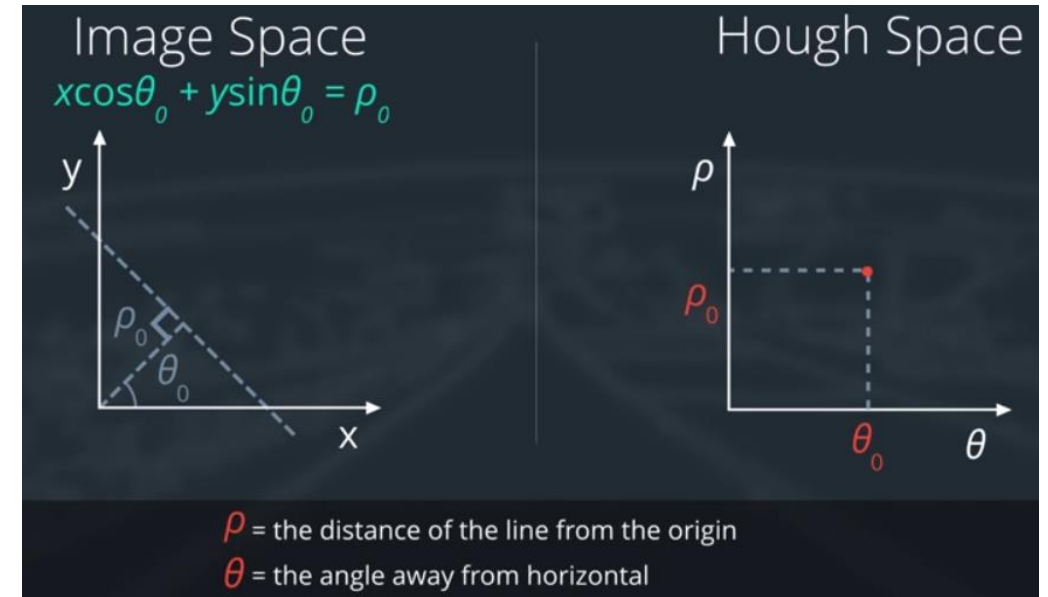
What does the intersection point of the two lines in Hough space correspond to in image space?

- ☒ A) A line in image space that passes through both (x_1, y_1) and (x_2, y_2)
- ☐ B) A line in image space that passes between (x_1, y_1) and (x_2, y_2)
- ☐ C) A line in image space that passes through (x_1, y_1)
- ☐ D) A line in image space that passes through only (x_2, y_2)

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.

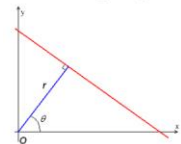
Theory [edit]

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^[6] proposed the use of the *Hesse normal form*

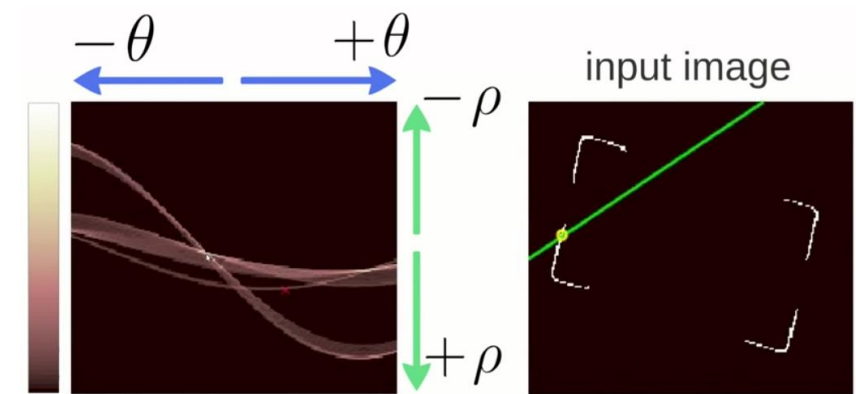
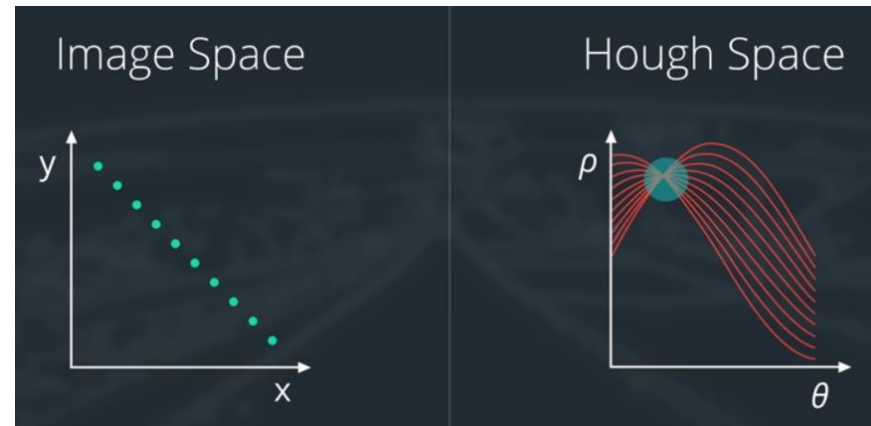
$$r = x \cos \theta + y \sin \theta,$$

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]



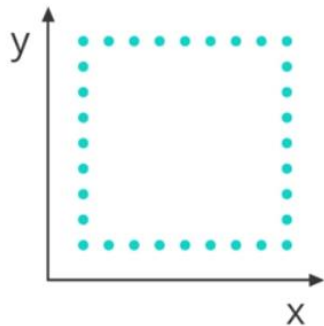
Other reference: [How Hough Transform works](https://en.wikipedia.org/wiki/Hough_transform)

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

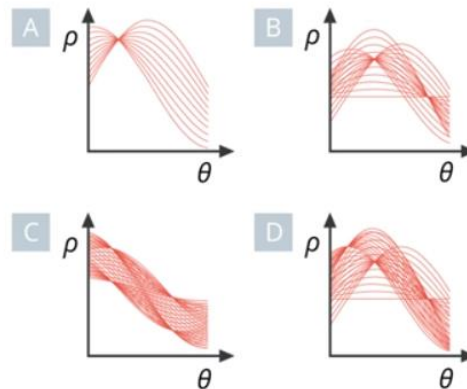
Hough Transform

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

Image Space



Hough Space



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**

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

In this case, we are operating on the image `masked_edges` (the output from Canny) and **the output from HoughLinesP will be lines**, which will simply be an array containing the endpoints (x1, y1, x2, y2) of all line segments detected by the transform operation. The other parameters define just what kind of line segments we're looking for.

First off, **rho** and **theta** are the distance and **angular resolution** of our grid in Hough space. Remember that, in Hough space, we have a grid laid out along the (Θ , ρ) axis. You need to specify rho in units of pixels and theta in units of radians.

So, what are reasonable values? Well, rho takes a minimum value of 1, and a reasonable starting place for theta is 1 degree ($\pi/180$ in radians). Scale these values up to be more flexible in your definition of what constitutes a line.

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 empty `np.array([])` is just a placeholder, no need to change it. `min_line_length` is the minimum length of a line (in pixels) that you will accept in (again, in pixels) between segments that you will allow to be cthe output, and `max_line_gap` is the maximum distance onnected into a single line. You can then iterate through your output lines and draw them onto the image to see what you got!



Code:
[05Hough_Transform_to_Find_Lane_Lines.txt](#)

```

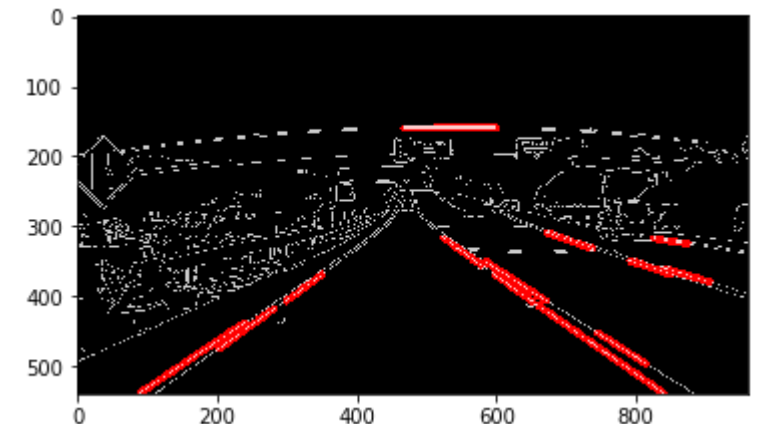
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import cv2 # Read in and grayscale the image
image = mpimg.imread('exit-ramp.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY) # Define a kernel size and apply Gaussian smoothing
kernel_size = 5
blur_gray = cv2.GaussianBlur(gray, (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)

```



Quiz: Hough Transform Quiz

Now it's your turn to play with the Hough Transform on an edge-detected image. You'll start with the image on the left below. If you "Test Run" the quiz, you'll get output that looks like the center image. Your job is to modify the parameters for the Hough Transform and impose a region of interest mask to get output that looks like the image on the right. In the code, I've given you a framework for defining a quadrilateral region of interest mask.



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

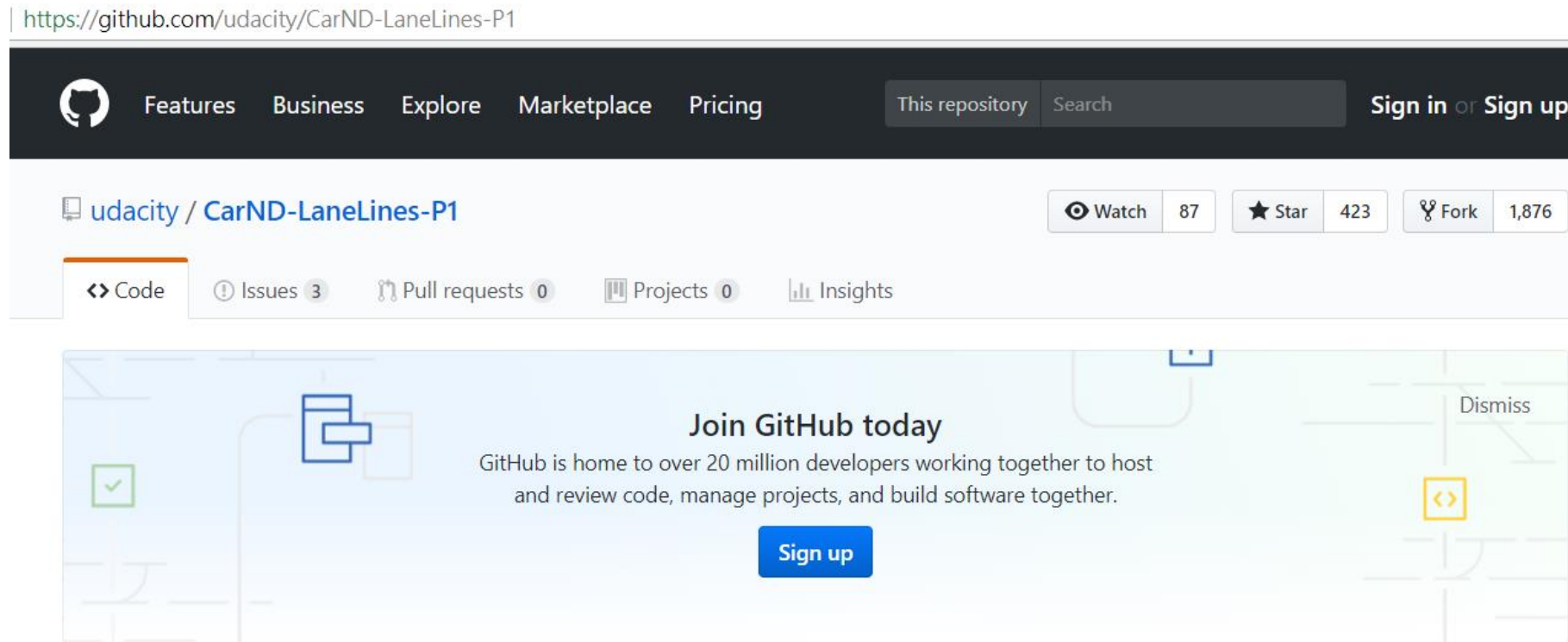


Canny Edge Detection & Hough Transform



Project Intro: Finding Lane Lines in a Video Stream

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



Lane Finding Project for Self-Driving Car ND

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

Lane Finding Project for Self-Driving Car ND

79 commits 2 branches 0 releases 18 contributors MIT

Branch: master New pull request Find file Clone or download

mvirgo Add license

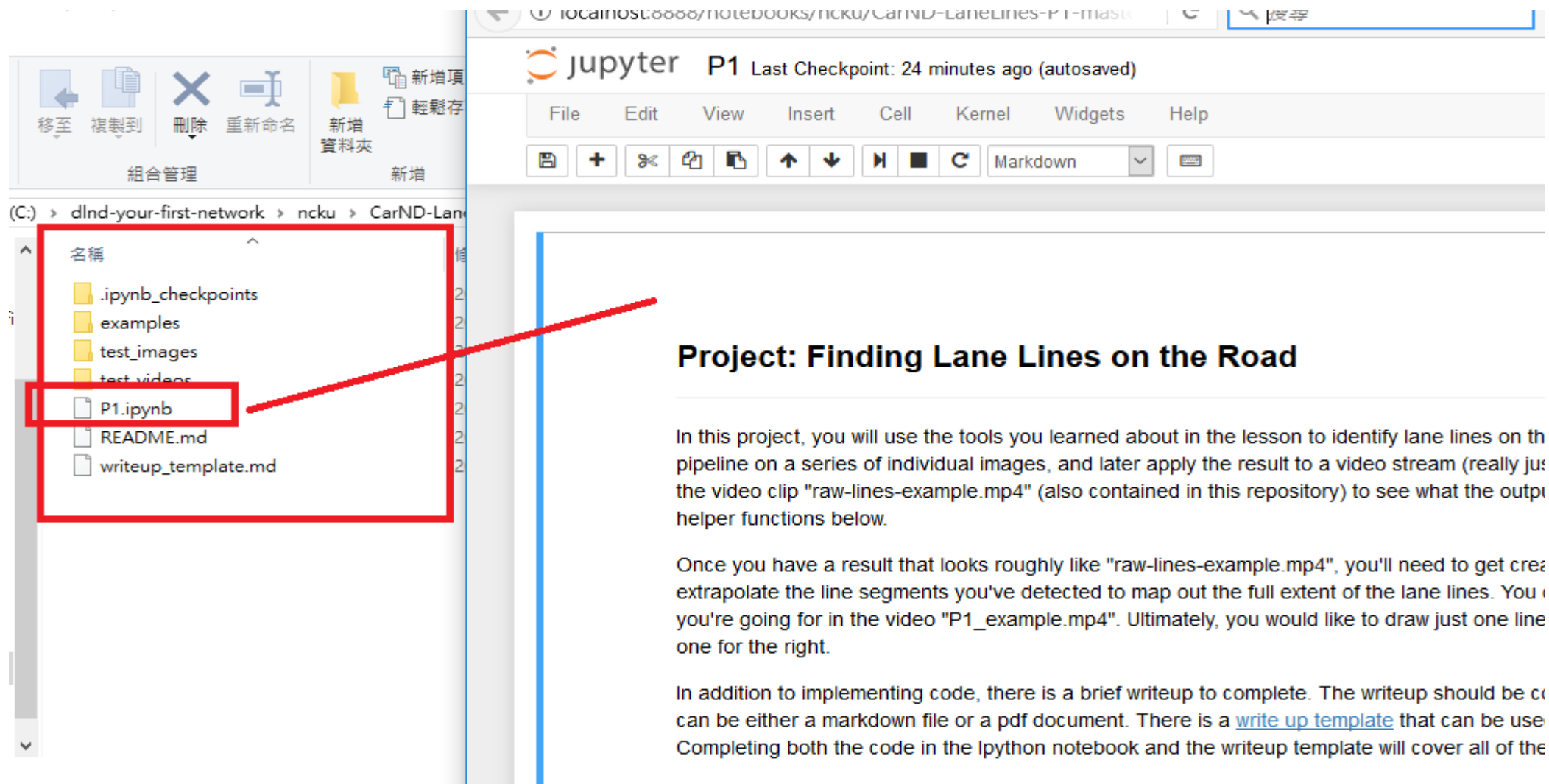
examples	move assets to appropriate folders
test_images	Add project files
test_videos	move assets to appropriate folders
LICENSE	Add license

Clone with HTTPS ⓘ
Use Git or checkout with SVN using the web URL.
<https://github.com/udacity/CarND-LaneLines>

Open in Desktop Download ZIP

examples	move assets to appropriate folders
test_images	Add project files
test_videos	move assets to appropriate folders
LICENSE	Add license
P1.ipynb	Update P1.ipynb
README.md	Update README.md
writup_template.md	Update writup_template.md

project



The screenshot displays a Jupyter Notebook environment. On the left, a file explorer shows the directory structure of a project. The files listed are:

- .ipynb_checkpoints
- examples
- test_images
- test_videos
- P1.ipynb
- README.md
- writeup_template.md

A red box highlights the **P1.ipynb** file, and a red arrow points from it to the notebook page. The notebook page has a title bar that reads "jupyter P1 Last Checkpoint: 24 minutes ago (autosaved)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The toolbar shows various icons for file operations and notebook navigation. The main content area of the notebook displays the following text:

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 the pipeline on a series of individual images, and later apply the result to a video stream (really just the video clip "raw-lines-example.mp4" (also contained in this repository) to see what the output helper functions below.

Once you have a result that looks roughly like "raw-lines-example.mp4", you'll need to get creative and extrapolate the line segments you've detected to map out the full extent of the lane lines. You're going for in the video "P1_example.mp4". Ultimately, you would like to draw just one line on the left and one for the right.

In addition to implementing code, there is a brief writeup to complete. The writeup should be completed and can be either a markdown file or a pdf document. There is a [write up template](#) that can be used. Completing both the code in the Ipython notebook and the writeup template will cover all of the

Install moviepy

```
In [6]: # Import everything needed to edit/save/watch video clips
from moviepy.editor import VideoFileClip
from IPython.display import HTML

-----
ImportError                                Traceback (most recent call last)
<ipython-input-6-b16dd2c24d5c> in <module>()
      1 # Import everything needed to edit/save/watch video clips
----> 2 from moviepy.editor import VideoFileClip
      3 from IPython.display import HTML

ImportError: No module named 'moviepy'
```

https://anaconda.org/search?q=moviepy

ANACONDA CLOUD

moviepy

Filters: Type: All Access: All Platform: All

Favorites	Downloads	Package (owner / package)	Platforms
0	1638	pypi / moviepy 0.21.8.12 Module for script-based video editing	source
0	1087	conda-forge / moviepy 0.2.3.2 Video editing with Python	linux-64 osx-64 win-32 win-64
0	173	jlaforet / moviepy 0.2.2.1 Video editing with Python	linux-32 linux-64 osx-64 win-32 win-64

https://anaconda.org/conda-forge/moviepy

Conda Files

License: MIT
Home: <https://zulko.github.io/moviepy>
Development: <https://github.com/Zulko/moviepy>
Documentation: <https://zulko.github.io/moviepy>
1088 total downloads

Installers

conda install ?

- linux-64 v0.2.3.2
- win-32 v0.2.3.2
- win-64 v0.2.3.2
- osx-64 v0.2.3.2

To install this package with conda run:
conda install -c conda-forge moviepy

```
conda install -c conda-forge moviepy
[ncku] C:\Users\FUDEY>conda install -c conda-forge moviepy
Fetching package metadata
Solving package specifications: .....
Package plan for installation in environment C:\Users\FUDEY\Anaconda3\envs\ncku:

The following packages will be downloaded:

package | build | size | channel
-----|-----|-----|-----
ffmpeg-3.2.4 | | 16.5 MB | conda-forge
decorator-4.0.11 | py35_0 | 14 KB | conda-forge
tqdm-4.11.2 | py35_0 | 90 KB | conda-forge
imageio-2.1.2 | py35_0 | 3.2 MB | conda-forge
moviepy-0.2.3.2 | py35_0 | 148 KB | conda-forge
Total: 20.0 MB

The following NEW packages will be INSTALLED:

ffmpeg: 3.2.4-1 conda-forge
imageio: 2.1.2-py35_0 conda-forge
moviepy: 0.2.3.2-py35_0 conda-forge
tqdm: 4.11.2-py35_0 conda-forge

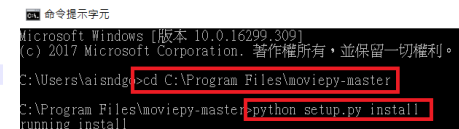
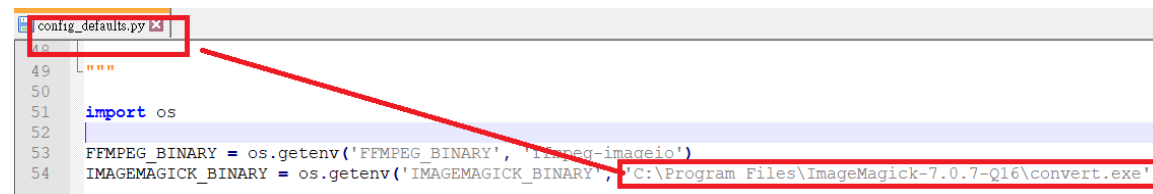
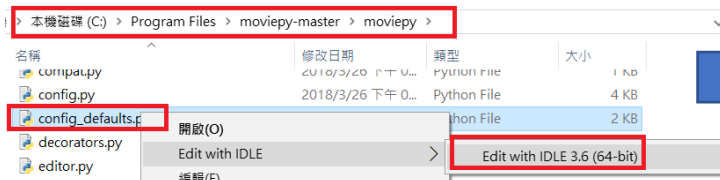
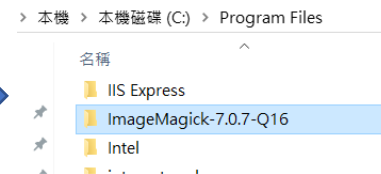
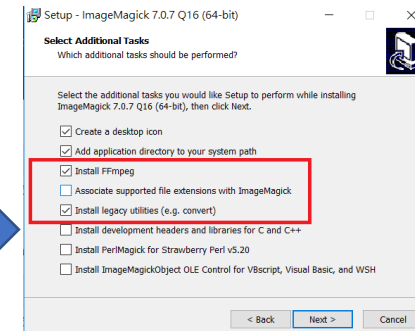
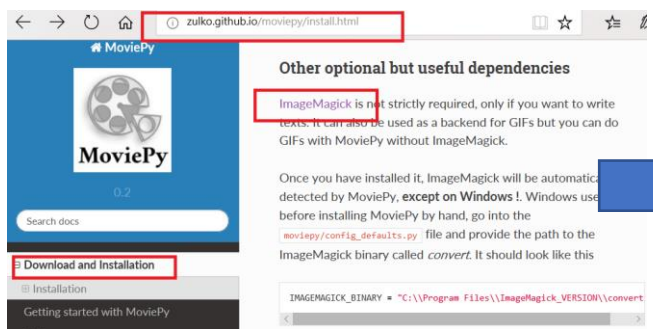
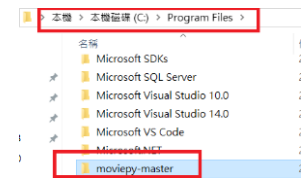
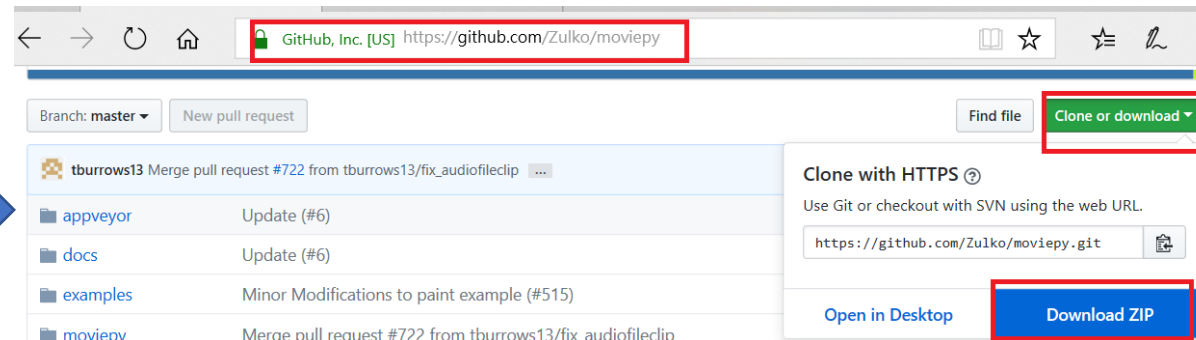
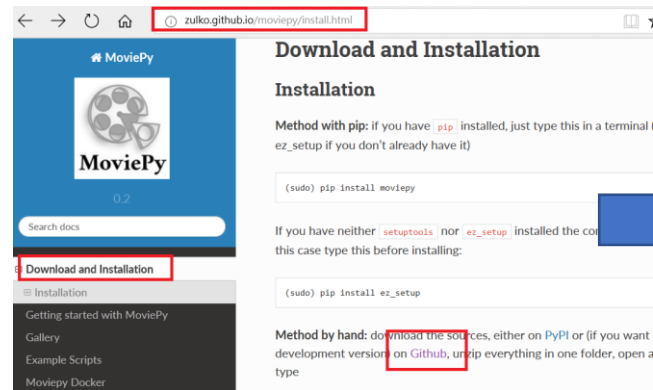
The following packages will be SUPERCEDED by a higher-priority channel:

decorator: 4.1.2-py35_0 --> 4.0.11-py35_0 conda-forge

Proceed ([y]/n)?
```

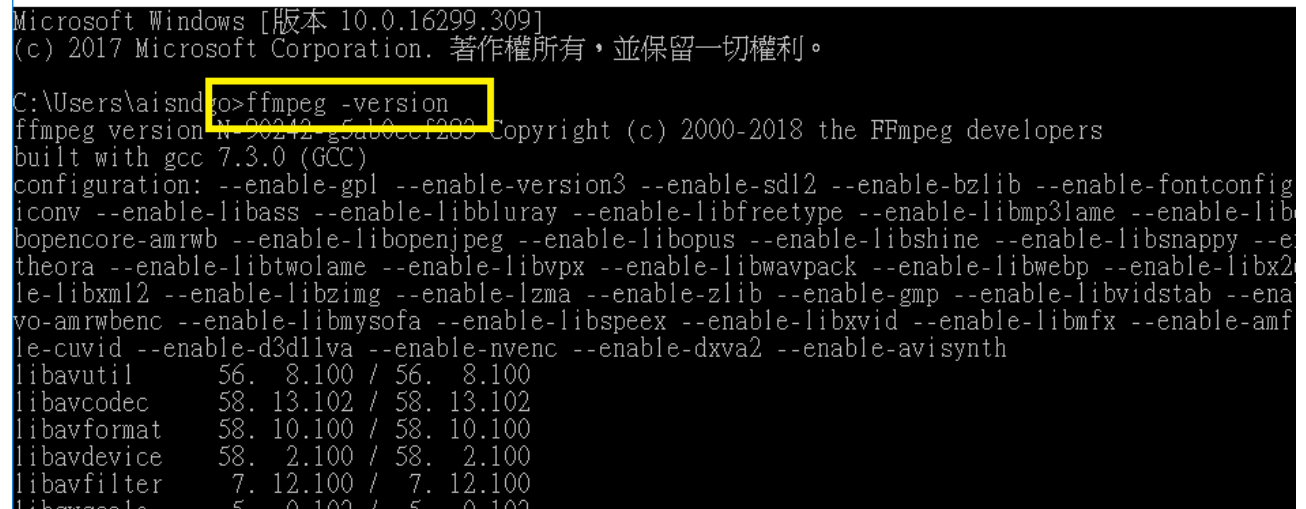
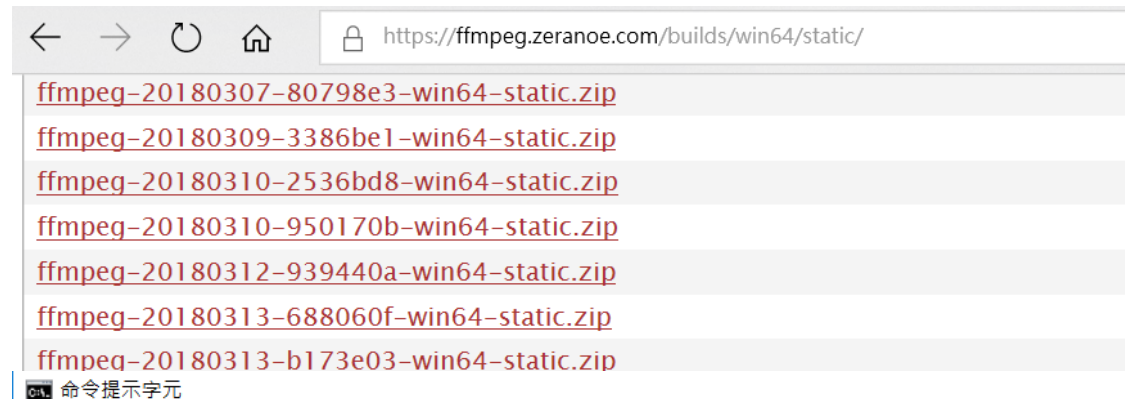
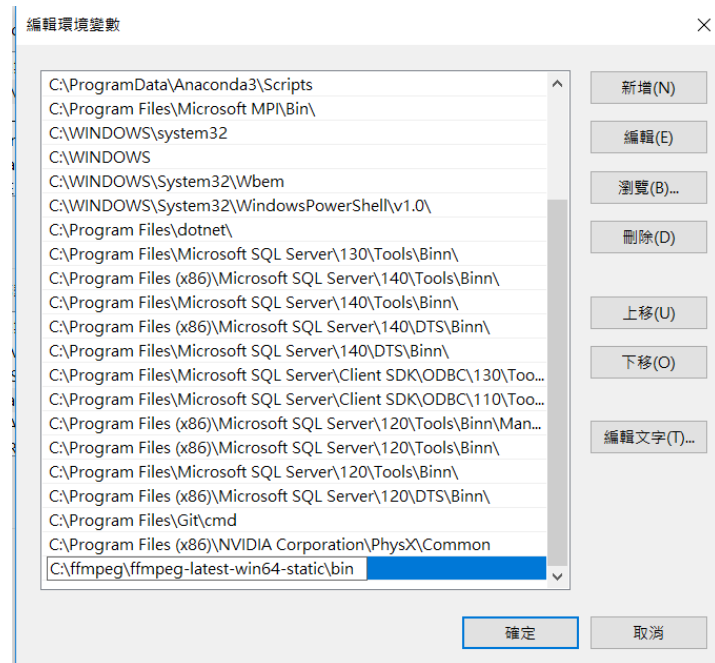
complete install Moviepy

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**, **grayscale**, **Gaussian smoothing**, **Canny Edge Detection** and **Hough Transform 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 gaussian 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

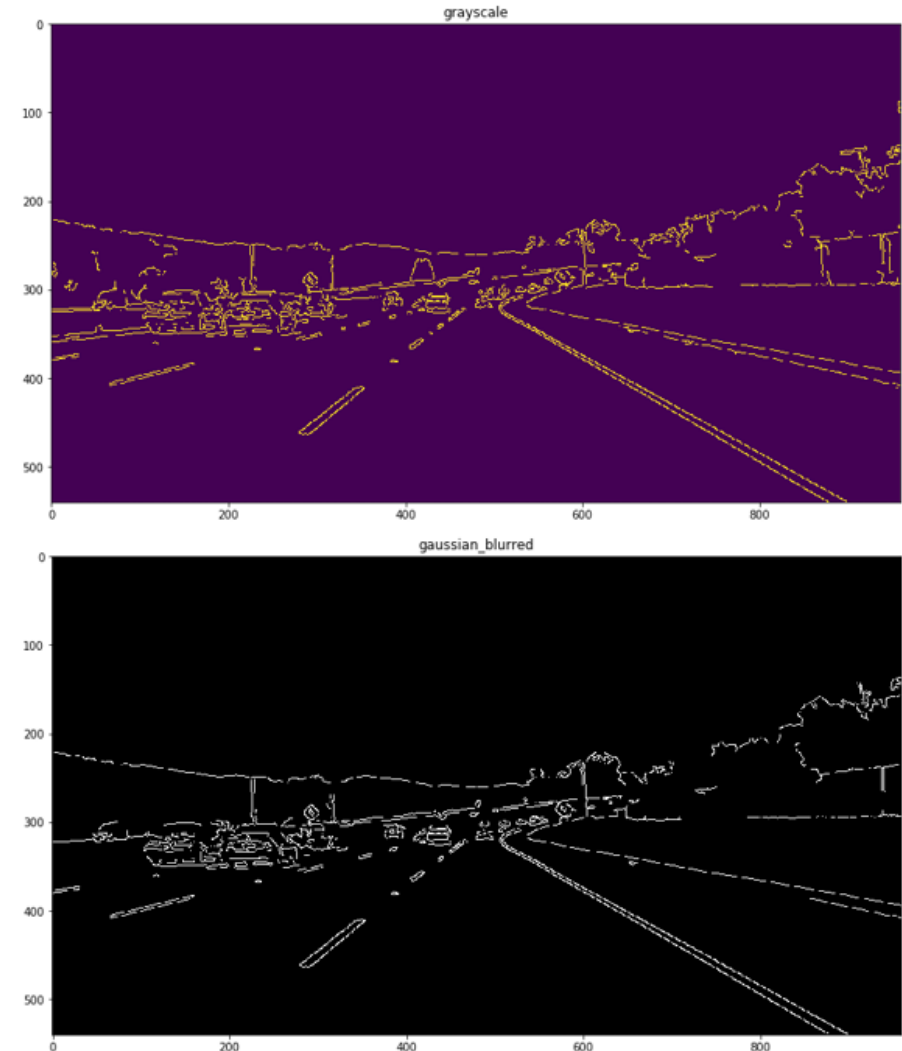
Optional Challenge



Test on Youtube Video

Canny edge detection and gaussian blur

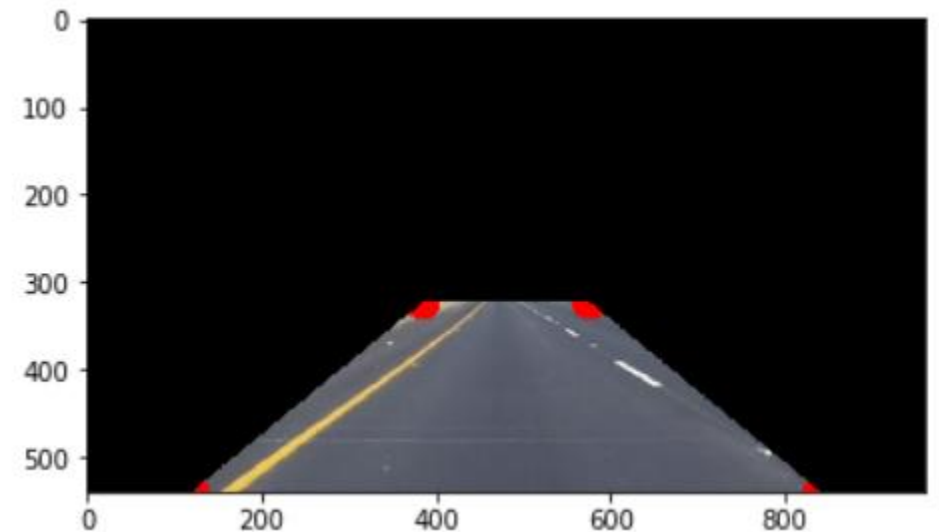
```
view_from_windshield = mpimg.imread('edge.jpg')
working_copy = np.copy(view_from_windshield)
# first convert to grayscale
grayscale = cv2.cvtColor(working_copy, cv2.COLOR_RGB2GRAY)
# apply gaussian blur before canny edge detection
kernel_size=3
gaussian_blurred = cv2.GaussianBlur(grayscale,(kernel_size, 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

```
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))
```



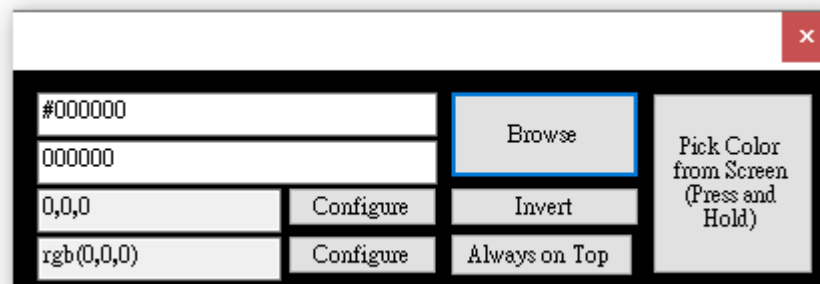
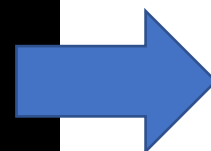
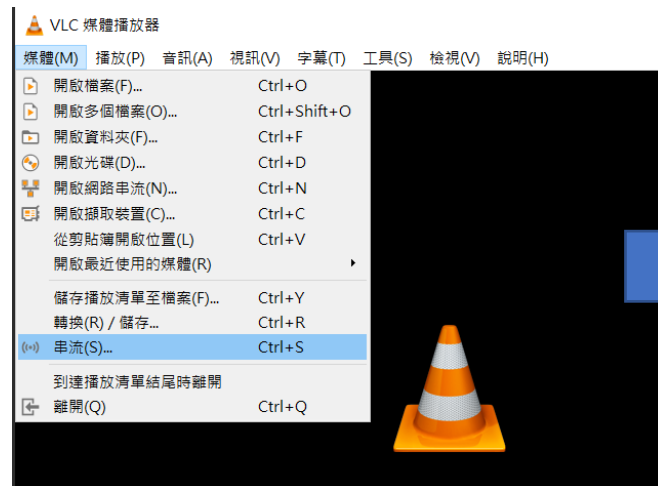
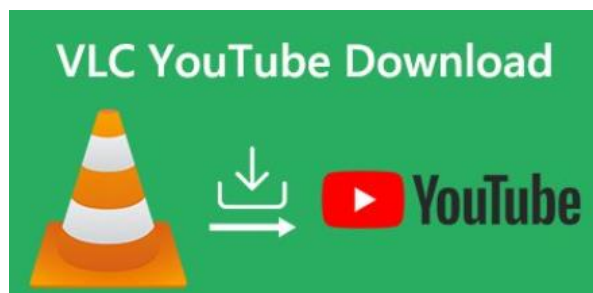
Test the function(detect Land Line)



<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 from 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 Lane & drawLine)
- Lane area drawing & Lane area drawing
- Determine the curvature of the lane and vehicle position with respect to center
- Test on Videos

