[..\02.【项目】Finding Lane Lines](../02.【项目】Finding Lane Lines)

[音频\pro2-小结.m4a](音频/pro2-小结.m4a)

|  |
| --- |
| 关键词：  image analysis  color channels  color seleciton   meet my color criterion  retained  blacked out  original image  color selection applied  Region Masking |
| [音频\pro2-01.m4a](音频/pro2-01.m4a)  color seleciton  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]  # 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)  # Uncomment the following code if you are running the code locally and wish to save the image  # mpimg.imsave("test-after.png", color\_select)  red\_threshold = green\_threshold = blue\_threshold = 200 |
| Region Masking  masking everything else out  **import** matplotlib.pyplot **as** plt**import** matplotlib.image **as** mpimg**import** numpy **as** np  *# Read in the image and print some stats*  image = mpimg.imread('test.jpg')  print('This image is: ', type(image),  'with dimensions:', image.shape)  *# Pull out the x and y sizes and make a copy of the image*  ysize = image.shape[0]  xsize = image.shape[1]  region\_select = np.copy(image)  *# Define a triangle region of interest # Keep in mind the origin (x=0, y=0) is in the upper left in image processing# Note: if you run this code, you'll find these are not sensible values!!# But you'll get a chance to play with them soon in a quiz*  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*  plt.imshow(region\_select)  6.Color and Region Combined  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)  line\_image = np.copy(image)  # Define color selection criteria  # MODIFY THESE VARIABLES TO MAKE YOUR COLOR SELECTION  red\_threshold = 200  green\_threshold = 200  blue\_threshold = 200  rgb\_threshold = [red\_threshold, green\_threshold, blue\_threshold]  # Define the vertices of a triangular mask.  # Keep in mind the origin (x=0, y=0) is in the upper left  # MODIFY THESE VALUES TO ISOLATE THE REGION  # WHERE THE LANE LINES ARE IN THE IMAGE  left\_bottom = [0, 539]  right\_bottom = [900, 300]  apex = [400, 0]  # Perform a linear fit (y=Ax+B) to each of the three sides of the triangle  # 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)  # Mask pixels below the threshold  color\_thresholds = (image[:,:,0] < rgb\_threshold[0]) | \  (image[:,:,1] < rgb\_threshold[1]) | \  (image[:,:,2] < rgb\_threshold[2])  # 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]))    # Mask color and region selection  color\_select[color\_thresholds | ~region\_thresholds] = [0, 0, 0]  # Color pixels red where both color and region selections met  line\_image[~color\_thresholds & region\_thresholds] = [255, 0, 0]  # Display the image and show region and color selections  plt.imshow(image)  x = [left\_bottom[0], right\_bottom[0], apex[0], left\_bottom[0]]  y = [left\_bottom[1], right\_bottom[1], apex[1], left\_bottom[1]]  plt.plot(x, y, 'b--', lw=4)  plt.imshow(color\_select)  plt.imshow(line\_image) |
| 8.Finding Lines of Any Color  9.What is Computer Vision    Canny Edge Detection  ------ |
| 10.Canny Edge Detection - 1  1）过程：先grayscale求出灰度图，再把灰度图里面的noise去掉一下。再求出gradient梯度图，最后在梯度图里找最强的，且相邻的点连起来  2）在Opencv里面如何调用edges方法    3）    4）导数的含义    5）openCV的Canny函数里是怎么确保找到的是最细的那个edge呢？  先是计算对X，对Y的导数找到gradient大的那些点，此时肯定边缘贼粗，因为边缘可能此时是好几行像素表示的。。。。。最高全找出---向外扩张到最低----- |
| 11.Canny to Detect Lane Lines  下面这些补充内容得去看  **[OpenCV Canny Docs](http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html" \t "D:/SDC/%E6%97%A0%E4%BA%BA%E9%A9%BE%E9%A9%B6%E5%B7%A5%E7%A8%8B%E5%B8%88%20%E7%AC%AC%E4%B8%80%E5%AD%A6%E6%9C%9F/02.%E3%80%90%E9%A1%B9%E7%9B%AE%E3%80%91Finding%20Lane%20Lines/_blank)** 记得去看   As far as a ratio of low\_threshold to high\_threshold, **[John Canny himself recommended](http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html" \l "steps" \t "D:/SDC/%E6%97%A0%E4%BA%BA%E9%A9%BE%E9%A9%B6%E5%B7%A5%E7%A8%8B%E5%B8%88%20%E7%AC%AC%E4%B8%80%E5%AD%A6%E6%9C%9F/02.%E3%80%90%E9%A1%B9%E7%9B%AE%E3%80%91Finding%20Lane%20Lines/_blank)** a low to high ratio of 1:2 or 1:3.  check out the **[OpenCV docs for GaussianBlur](http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=gaussianblur" \l "gaussianblur" \t "D:/SDC/%E6%97%A0%E4%BA%BA%E9%A9%BE%E9%A9%B6%E5%B7%A5%E7%A8%8B%E5%B8%88%20%E7%AC%AC%E4%B8%80%E5%AD%A6%E6%9C%9F/02.%E3%80%90%E9%A1%B9%E7%9B%AE%E3%80%91Finding%20Lane%20Lines/_blank)**   If you would like to dive into the math underpinning these functions, please check out the free Udacity course, **[Intro to Computer Vision](https://www.udacity.com/course/introduction-to-computer-vision--ud810" \t "D:/SDC/%E6%97%A0%E4%BA%BA%E9%A9%BE%E9%A9%B6%E5%B7%A5%E7%A8%8B%E5%B8%88%20%E7%AC%AC%E4%B8%80%E5%AD%A6%E6%9C%9F/02.%E3%80%90%E9%A1%B9%E7%9B%AE%E3%80%91Finding%20Lane%20Lines/_blank)**, |
| 12.Canny Edges  # Do 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  # Note: in the previous example we were reading a .jpg  # Here we read a .png and convert to 0,255 bytescale  image = mpimg.imread('exit-ramp.jpg')  gray = cv2.cvtColor(image,cv2.COLOR\_RGB2GRAY)  plt.imshow(gray, cmap='gray')  # Define a kernel size for Gaussian smoothing / blurring  suppressing noise and spurious gradients by averaging  kernel\_size = 3 # Must be an odd number (3, 5, 7...)  blur\_gray = cv2.GaussianBlur(gray,(kernel\_size, kernel\_size),0)  # Define our parameters for Canny and run it  low\_threshold = 1  high\_threshold = 10  edges = cv2.Canny(blur\_gray, low\_threshold, high\_threshold)  # Display the image  plt.imshow(edges, cmap='Greys\_r') |
| [音频\pro2-02.m4a](音频/pro2-02.m4a)  13.Hough Transform - 1  A line in Cartesian form |
| **angle-distance parameter space**     1）什么样的图片才可以作为hough transform 的输入？  2）fit 拟合  3）parameter space和hough space是一个意思，指同一个东西  4）介绍了第二种hough space是为了解决vertical line的问题 |
| 14.Hough Transform to Find Lane Lines  [音频\pro2-03.m4a](音频/pro2-03.m4a)   what kind of lines we want to detect (i.e., long lines, short lines, bendy lines, dashed lines, etc.).  OpenCV function called HoughLinesP  <https://alyssaq.github.io/2014/understanding-hough-transform/>    1）Given a set of edge points or a binary image indicating edges  receive. **2）Algorithm steps**   3） <https://github.com/alyssaq/hough_transform>   |  |  | | --- | --- | | import numpy as np  import imageio  import math  def rgb2gray(rgb):  return np.dot(rgb[..., :3], [0.299, 0.587, 0.114]).astype(np.uint8)  def hough\_line(img, angle\_step=1, lines\_are\_white=True, value\_threshold=5):  """  Hough transform for lines  Input:  img - 2D binary image with nonzeros representing edges  angle\_step - Spacing between angles to use every n-th angle  between -90 and 90 degrees. Default step is 1.  lines\_are\_white - boolean indicating whether lines to be detected are white  value\_threshold - Pixel values above or below the value\_threshold are edges  Returns:  accumulator - 2D array of the hough transform accumulator  theta - array of angles used in computation, in radians.  rhos - array of rho values. Max size is 2 times the diagonal  distance of the input image.  """  # Rho and Theta ranges  thetas = np.deg2rad(np.arange(-90.0, 90.0, angle\_step))  width, height = img.shape  diag\_len = int(round(math.sqrt(width \* width + height \* height)))  rhos = np.linspace(-diag\_len, diag\_len, diag\_len \* 2)  # Cache some resuable values  cos\_t = np.cos(thetas)  sin\_t = np.sin(thetas)  num\_thetas = len(thetas)  # Hough accumulator array of theta vs rho  accumulator = np.zeros((2 \* diag\_len, num\_thetas), dtype=np.uint8)  # (row, col) indexes to edges  are\_edges = img > value\_threshold if lines\_are\_white else img < value\_threshold  y\_idxs, x\_idxs = np.nonzero(are\_edges)  # Vote in the hough accumulator  for i in range(len(x\_idxs)):  x = x\_idxs[i]  y = y\_idxs[i] | for t\_idx in range(num\_thetas):  # Calculate rho. diag\_len is added for a positive index  rho = diag\_len + int(round(x \* cos\_t[t\_idx] + y \* sin\_t[t\_idx]))  accumulator[rho, t\_idx] += 1  return accumulator, thetas, rhos  def show\_hough\_line(img, accumulator, thetas, rhos, save\_path=None):  import matplotlib.pyplot as plt  fig, ax = plt.subplots(1, 2, figsize=(10, 10))  ax[0].imshow(img, cmap=plt.cm.gray)  ax[0].set\_title('Input image')  ax[0].axis('image')  ax[1].imshow(  accumulator, cmap='jet',  extent=[np.rad2deg(thetas[-1]), np.rad2deg(thetas[0]), rhos[-1], rhos[0]])  ax[1].set\_aspect('equal', adjustable='box')  ax[1].set\_title('Hough transform')  ax[1].set\_xlabel('Angles (degrees)')  ax[1].set\_ylabel('Distance (pixels)')  ax[1].axis('image')  # plt.axis('off')  if save\_path is not None:  plt.savefig(save\_path, bbox\_inches='tight')  plt.show()  if \_\_name\_\_ == '\_\_main\_\_':  imgpath = 'imgs/binary\_crosses.png'  img = imageio.imread(imgpath)  if img.ndim == 3:  img = rgb2gray(img)  accumulator, thetas, rhos = hough\_line(img)  show\_hough\_line(img, accumulator, save\_path='imgs/output.png') |     4）例子    Hough transform (and the faster probabilistic version) is available in [openCV](http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html" \t "https://alyssaq.github.io/2014/understanding-hough-transform/_blank) and [scikit-image](http://scikit-image.org/docs/dev/auto_examples/plot_line_hough_transform.html" \t "https://alyssaq.github.io/2014/understanding-hough-transform/_blank).  再出现说一个空间向另一个空间转换的问题该怎么思考？1、纯代数计算；2、将新的空间做成array，并一点点增加值  np.argmax什么意思 返回最大值得索引  np.nonzero(img)什么意思 用于得到数组array中非零元素的位置（数组索引）的函数。  5）      我在想能不能今天做个GitHub的提交来实现这三个不同的边缘检测  6）    7）      8）     |  |  | | --- | --- | | # Do relevant imports  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  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  theta = np.pi/180  threshold = 1  min\_line\_length = 10  max\_line\_gap = 1  line\_image = np.copy(image)\*0 #creating a blank to draw lines on  # Run Hough on edge detected image  lines = cv2.HoughLinesP(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((edges, edges, edges))  # Draw the lines on the edge image  combo = cv2.addWeighted(color\_edges, 0.8, line\_image, 1, 0)  plt.imshow(combo) |   9） |
| 15.Hough Transform [音频\pro2-04.m4a](音频/pro2-04.m4a)  1) 练习: Hough Transform Quiz    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  edges = cv2.Canny(blur\_gray, low\_threshold, high\_threshold)  # Next we'll create a masked edges image using cv2.fillPoly()  mask = np.zeros\_like(edges)  ignore\_mask\_color = 255  # This time we are defining a four sided polygon to mask  imshape = image.shape  ~~vertices = np.array([[(0,imshape[0]),(0, 0), (imshape[1], 0), (imshape[1],imshape[0])]], dtype=np.int32)~~  vertices = np.array([[(0,imshape[0]),(450, 290), (490, 290), (imshape[1],imshape[0])]], dtype=np.int32)  cv2.fillPoly(mask, vertices, ignore\_mask\_color)  masked\_edges = cv2.bitwise\_and(edges, mask)  # Define the Hough transform parameters  # Make a blank the same size as our image to draw on  rho = ~~1~~ 2# distance resolution in pixels of the Hough grid  theta = np.pi/180 # angular resolution in radians of the Hough grid  threshold = 1 # minimum number of votes (intersections in Hough grid cell)  min\_line\_length = 5 #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  # Output "lines" is an array containing endpoints of detected line segments  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 a blank image  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((edges, edges, edges))  # Draw the lines on the edge image  lines\_edges = cv2.addWeighted(color\_edges, 0.8, line\_image, 1, 0)  plt.imshow(lines\_edges) |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| 小结：  color selection  Region Masking又叫region of interest  grayscale  GaussianBlur  Cv2.Canny |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |