****

计算机视觉实验

**实验四**

**专 业：**

**班 级：**

**姓 名：**

**学 号：**

**指导教师：**

**成 绩：**

**计算机学院**

**年 月 日**

# 实验目的

纹理图像的共生矩阵计算

找纹理图像（也可以是自己的毛衣、桌面等）

计算共生矩阵及相应的一些指标，进一步掌握相关理论。

Snake和hough变换

根据代码掌握相关理论

# 实验过程

## 纹理图像的共生矩阵计算

实验图片



所用代码

import numpy as np

import cv2

from skimage.feature import greycomatrix, greycoprops

# 1. 读取图像文件

image = cv2.imread('in.jpg', cv2.IMREAD\_GRAYSCALE)

# 2. 计算灰度共生矩阵

angle = np.pi

matrix\_coocurrence = greycomatrix(image, [1], [angle], levels=256, normed=False, symmetric=False)

# 3. 计算统计值

contrast = greycoprops(matrix\_coocurrence, 'contrast')

dissimilarity = greycoprops(matrix\_coocurrence, 'dissimilarity')

homogeneity = greycoprops(matrix\_coocurrence, 'homogeneity')

energy = greycoprops(matrix\_coocurrence, 'energy')

correlation = greycoprops(matrix\_coocurrence, 'correlation')

asm = greycoprops(matrix\_coocurrence, 'ASM')

# 4. 显示结果

print('对比度 = {}'.format(contrast))

print('相异性 = {}'.format(dissimilarity))

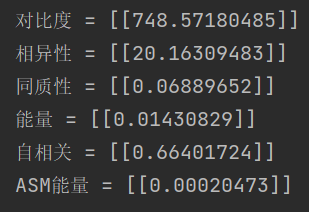
print('同质性 = {}'.format(homogeneity))

print('能量 = {}'.format(energy))

print('自相关 = {}'.format(correlation))

print('ASM能量 = {}'.format(asm))

实验效果

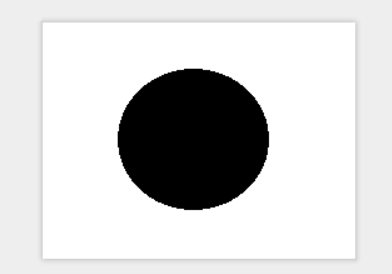


代码分析及现象

灰度共生矩阵法(GLCM, Gray-level co-occurrence matrix)通过计算灰度图像得到它的共生矩阵，然后再计算所得到的共生矩阵的部分特征值，来分别代表图像的某些纹理特征。灰度共生矩阵能反映图像灰度关于方向、相邻间隔、变化幅度等综合信息，它是分析图像的局部模式和它们排列规则的基础。当我们计算得到共生矩阵时利用这个共生矩阵，我们可以计算出很多种特征值来表示其纹理特性，一般比较常用的有角二阶矩（能量），对比度，相关度，熵等。

## Snake算法

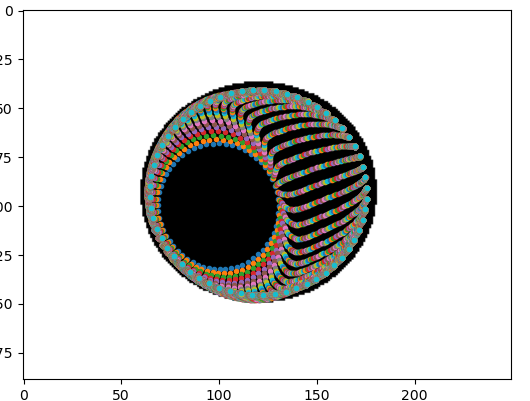
实验图片



所用代码

|  |
| --- |
| snake.py |
| import numpy as np import cv2 import matplotlib.pyplot as plt  def snake(img, x, y, alpha=0.001, beta=0.4, gamma=100,  sigma=20, iterations=500):  """  The snake algorithm to segment image   Parameters  ------  img : ndarray  input image   ------  x,y : ndarry  X-coordinate and Y-coordinate of the initial contour  alpha,beta: number  The set of parameters of internal energy  gamma : number  Parameter cotrolling the external engery  sigma : number  Standard deviation  iterations : number  The number of iteration  """  # points\_x\_y = []  fig = plt.figure()  plt.imshow(img, cmap='gray')  # points\_x\_y.append(plt.plot(x, y, '.'))  # compute the matrix  N = np.size(x)  a = gamma \* (2 \* alpha + 6 \* beta) + 1  b = gamma \* (-alpha - 4 \* beta)  c = gamma \* beta  p = np.zeros((N, N), dtype=np.float)  p[0] = np.c\_[a, b, c, np.zeros((1, N - 5)), c, b]  for i in range(N):  p[i] = np.roll(p[0], i)  p = np.linalg.inv(p)  # filter the image  smoothed = cv2.GaussianBlur((img - img.min()) / (img.max() - img.min()), (89, 89), sigma)  giy, gix = np.gradient(smoothed)  gmi = (gix \*\* 2 + giy \*\* 2) \*\* 0.5  gmi = (gmi - gmi.min()) / (gmi.max() - gmi.min())  Iy, Ix = np.gradient(gmi)   # avoid the curvature evolve to the outside of the image  def fmax(x, y):  x[x < 0] = 0  y[y < 0] = 0  x[x > img.shape[1] - 1] = img.shape[1] - 1  y[y > img.shape[0] - 1] = img.shape[0] - 1  return y.round().astype(int), x.round().astype(int)   for i in range(iterations):  fex = Ix[fmax(x, y)]  fey = Iy[fmax(x, y)]  x = np.dot(p, x + gamma \* fex)  y = np.dot(p, y + gamma \* fey)  plt.plot(x, y, '.')  # points\_x\_y.append(plt.plot(x, y, '.'))  # ani = animation.ArtistAnimation(fig, points\_x\_y, interval=200, blit=True, repeat=True)  plt.show()  # ani.save('1.gif', fps=30)  return x, y |
| main.py |
| import numpy as np import cv2 from snake import \*  #read the image Image = cv2.imread('example.bmp',1) image = cv2.cvtColor(Image,cv2.COLOR\_BGR2GRAY) img = np.array(image,dtype = np.float)  #define the initial snake t = np.linspace(0,2\*np.pi,60,endpoint = True) x\_0 = 100+30\*np.sin(t) y\_0 = 100+30\*np.cos(t)  #plot the image and results x\_1,y\_1= snake(img,x\_0,y\_0) |

实验效果



代码分析及现象

Snake模型基本原理是表征拟合误差的“能量”为最小化的曲线.设对于拟合目标有一个待选曲线集,定义能量函数与待选集中每一条曲线相关联,能量函数的设计原则就是:有利属性要能导致能量缩小。有利属性包括:曲线连续、平滑、曲线与高梯度区域接近以及其他一些具体的先验知识。活动轮廓在取值范围内移动时,就能在能量函数的指导下收敛到局部边界,且能保持曲线的连续和平滑。

简单来讲，Snake模型就是一条可变形的参数曲线及相应的能量函数，以最小化能量目标函数为目标，控制参数曲线变形，具有最小能量的闭合曲线就是目标轮廓。

### Hough算法

实验图片

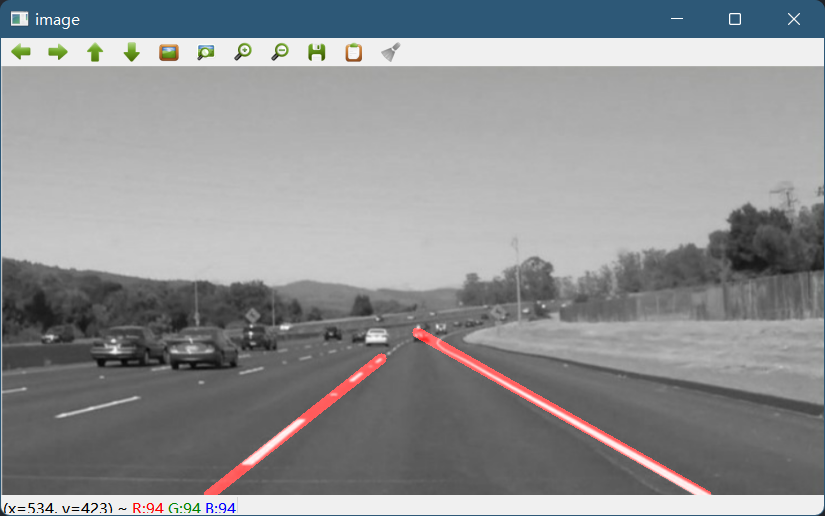




所用代码

|  |
| --- |
| lane\_detect.py |
| import cv2 import numpy as np   def grayscale(image):  return cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)   def canny(image, low\_threshold, high\_threshold):  return cv2.Canny(image, low\_threshold, high\_threshold)   def gaussian\_blur(image, k):  return cv2.GaussianBlur(image, (k, k), 0)   def region\_of\_interest(image, vertices):  mask = np.zeros\_like(image) # 生成图像大小一致的zeros矩阵   # 填充顶点vertices中间区域  if len(image.shape) > 2:  channel\_count = image.shape[2]  ignore\_mask\_color = (255,) \* channel\_count  else:  ignore\_mask\_color = 255   # 填充函数  cv2.fillPoly(mask, vertices, ignore\_mask\_color)  masked\_image = cv2.bitwise\_and(image, mask)  return masked\_image   def weighted\_image(img, initial\_img, a=0.8, b=1.0, c=0.):  return cv2.addWeighted(initial\_img, a, img, b, c)   def hough\_lines(img, rho, theta, threshold, min\_line\_len, max\_line\_gap):  """  :param img:  :param rho: 线段以像素为单位的距离精度  :param theta: 像素以弧度为单位的角度精度(np.pi/180较为合适)  :param threshold: 霍夫平面累加的阈值  :param min\_line\_len: 线段最小长度(像素级)  :param max\_line\_gap: 最大允许断裂长度  :return:  """  lines = cv2.HoughLinesP(img, rho, theta, threshold, np.array([]), min\_line\_len, max\_line\_gap)  return lines   def draw\_lines(image, lines, color=[0, 0, 255], thickness=2):  right\_y\_set = []  right\_x\_set = []  right\_slope\_set = []   left\_y\_set = []  left\_x\_set = []  left\_slope\_set = []   slope\_min = .35 # 斜率低阈值  slope\_max = .85 # 斜率高阈值  middle\_x = image.shape[1] / 2 # 图像中线x坐标  max\_y = image.shape[0] # 最大y坐标   for line in lines:  for x1, y1, x2, y2 in line:  fit = np.polyfit((x1, x2), (y1, y2), 1) # 拟合成直线  slope = fit[0] # 斜率   if slope\_min < np.absolute(slope) <= slope\_max:   # 将斜率大于0且线段X坐标在图像中线右边的点存为右边车道线  if slope > 0 and x1 > middle\_x and x2 > middle\_x:  right\_y\_set.append(y1)  right\_y\_set.append(y2)  right\_x\_set.append(x1)  right\_x\_set.append(x2)  right\_slope\_set.append(slope)   # 将斜率小于0且线段X坐标在图像中线左边的点存为左边车道线  elif slope < 0 and x1 < middle\_x and x2 < middle\_x:  left\_y\_set.append(y1)  left\_y\_set.append(y2)  left\_x\_set.append(x1)  left\_x\_set.append(x2)  left\_slope\_set.append(slope)   # 绘制左车道线  if left\_y\_set:  lindex = left\_y\_set.index(min(left\_y\_set)) # 最高点  left\_x\_top = left\_x\_set[lindex]  left\_y\_top = left\_y\_set[lindex]  lslope = np.median(left\_slope\_set) # 计算平均值   # 根据斜率计算车道线与图片下方交点作为起点  left\_x\_bottom = int(left\_x\_top + (max\_y - left\_y\_top) / lslope)   # 绘制线段  cv2.line(image, (left\_x\_bottom, max\_y), (left\_x\_top, left\_y\_top), color, thickness)   # 绘制右车道线  if right\_y\_set:  rindex = right\_y\_set.index(min(right\_y\_set)) # 最高点  right\_x\_top = right\_x\_set[rindex]  right\_y\_top = right\_y\_set[rindex]  rslope = np.median(right\_slope\_set)   # 根据斜率计算车道线与图片下方交点作为起点  right\_x\_bottom = int(right\_x\_top + (max\_y - right\_y\_top) / rslope)   # 绘制线段  cv2.line(image, (right\_x\_top, right\_y\_top), (right\_x\_bottom, max\_y), color, thickness)   def lane\_detect(image):  rho = 1 # 霍夫像素单位  theta = np.pi / 180 # 霍夫角度移动步长  hof\_threshold = 20 # 霍夫平面累加阈值threshold  min\_line\_len = 30 # 线段最小长度  max\_line\_gap = 60 # 最大允许断裂长度   kernel\_size = 5 # 高斯滤波器大小size  canny\_low\_threshold = 75 # canny边缘检测低阈值  canny\_high\_threshold = canny\_low\_threshold \* 3 # canny边缘检测高阈值   alpha = 0.8 # 原图像权重  beta = 1. # 车道线图像权重  lambda\_ = 0.   imshape = image.shape # 获取图像大小   # 灰度图转换  gray = grayscale(image)   # 高斯滤波  blur\_gray = gaussian\_blur(gray, kernel\_size)   # Canny边缘检测  edge\_image = canny(blur\_gray, canny\_low\_threshold, canny\_high\_threshold)   # 生成Mask掩模  vertices = np.array([[(0, imshape[0]), (9 \* imshape[1] / 20, 11 \* imshape[0] / 18),  (11 \* imshape[1] / 20, 11 \* imshape[0] / 18), (imshape[1], imshape[0])]], dtype=np.int32)  masked\_edges = region\_of\_interest(edge\_image, vertices)   # 基于霍夫变换的直线检测  lines = hough\_lines(masked\_edges, rho, theta, hof\_threshold, min\_line\_len, max\_line\_gap)  line\_image = np.zeros\_like(image)   # 绘制车道线线段  draw\_lines(line\_image, lines, thickness=10)   # 图像融合  lines\_edges = weighted\_image(image, line\_image, alpha, beta, lambda\_)  return lines\_edges   def img\_detect\_demo():  img = cv2.imread('lane.jpg')  out = lane\_detect(img)  cv2.imshow("image", out)  cv2.waitKey()   def video\_detect\_demo():  cap = cv2.VideoCapture("project\_video.mp4")  count = 0  while(cap.isOpened()):  ret\_value, frame = cap.read()  if ret\_value == 0:  break  processed = lane\_detect(frame)  cv2.imshow("image", processed)  count += 1  print(count)  if cv2.waitKey(10) & 0xFF == ord('q'):  break  cap.release()  cv2.destroyAllWindows()   if \_\_name\_\_ == '\_\_main\_\_':  img\_detect\_demo()  # video\_detect\_demo() |
| plate\_detect.py |
| import cv2 import numpy as np   def grayscale(image):  return cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)   def canny(image, low\_threshold, high\_threshold):  return cv2.Canny(image, low\_threshold, high\_threshold)   def gaussian\_blur(image, k):  return cv2.GaussianBlur(image, (k, k), 0)   def region\_of\_interest(image, vertices):  mask = np.zeros\_like(image) # 生成图像大小一致的zeros矩阵   # 填充顶点vertices中间区域  if len(image.shape) > 2:  channel\_count = image.shape[2]  ignore\_mask\_color = (255,) \* channel\_count  else:  ignore\_mask\_color = 255   # 填充函数  cv2.fillPoly(mask, vertices, ignore\_mask\_color)  masked\_image = cv2.bitwise\_and(image, mask)  return masked\_image   def weighted\_image(img, initial\_img, a=0.8, b=1.0, c=0.):  return cv2.addWeighted(initial\_img, a, img, b, c)   def hough\_lines(img, rho, theta, threshold, min\_line\_len, max\_line\_gap):  """  :param img:  :param rho: 线段以像素为单位的距离精度  :param theta: 像素以弧度为单位的角度精度(np.pi/180较为合适)  :param threshold: 霍夫平面累加的阈值  :param min\_line\_len: 线段最小长度(像素级)  :param max\_line\_gap: 最大允许断裂长度  :return:  """  lines = cv2.HoughLinesP(img, rho, theta, threshold, np.array([]), min\_line\_len, max\_line\_gap)  return lines   def draw\_lines(image, lines, color=[0, 0, 255], thickness=2):  right\_y\_set = []  right\_x\_set = []  right\_slope\_set = []   left\_y\_set = []  left\_x\_set = []  left\_slope\_set = []   slope\_min = .35 # 斜率低阈值  slope\_max = .85 # 斜率高阈值  middle\_x = image.shape[1] / 2 # 图像中线x坐标  max\_y = image.shape[0] # 最大y坐标   for line in lines:  for x1, y1, x2, y2 in line:  fit = np.polyfit((x1, x2), (y1, y2), 1) # 拟合成直线  slope = fit[0] # 斜率   if slope\_min < np.absolute(slope) <= slope\_max:   # 将斜率大于0且线段X坐标在图像中线右边的点存为右边车道线  if slope > 0 and x1 > middle\_x and x2 > middle\_x:  right\_y\_set.append(y1)  right\_y\_set.append(y2)  right\_x\_set.append(x1)  right\_x\_set.append(x2)  right\_slope\_set.append(slope)   # 将斜率小于0且线段X坐标在图像中线左边的点存为左边车道线  elif slope < 0 and x1 < middle\_x and x2 < middle\_x:  left\_y\_set.append(y1)  left\_y\_set.append(y2)  left\_x\_set.append(x1)  left\_x\_set.append(x2)  left\_slope\_set.append(slope)   # 绘制左车道线  if left\_y\_set:  lindex = left\_y\_set.index(min(left\_y\_set)) # 最高点  left\_x\_top = left\_x\_set[lindex]  left\_y\_top = left\_y\_set[lindex]  lslope = np.median(left\_slope\_set) # 计算平均值   # 根据斜率计算车道线与图片下方交点作为起点  left\_x\_bottom = int(left\_x\_top + (max\_y - left\_y\_top) / lslope)   # 绘制线段  cv2.line(image, (left\_x\_bottom, max\_y), (left\_x\_top, left\_y\_top), color, thickness)   # 绘制右车道线  if right\_y\_set:  rindex = right\_y\_set.index(min(right\_y\_set)) # 最高点  right\_x\_top = right\_x\_set[rindex]  right\_y\_top = right\_y\_set[rindex]  rslope = np.median(right\_slope\_set)   # 根据斜率计算车道线与图片下方交点作为起点  right\_x\_bottom = int(right\_x\_top + (max\_y - right\_y\_top) / rslope)   # 绘制线段  cv2.line(image, (right\_x\_top, right\_y\_top), (right\_x\_bottom, max\_y), color, thickness)   def lane\_detect(image):  rho = 1 # 霍夫像素单位  theta = np.pi / 180 # 霍夫角度移动步长  hof\_threshold = 20 # 霍夫平面累加阈值threshold  min\_line\_len = 30 # 线段最小长度  max\_line\_gap = 60 # 最大允许断裂长度   kernel\_size = 5 # 高斯滤波器大小size  canny\_low\_threshold = 75 # canny边缘检测低阈值  canny\_high\_threshold = canny\_low\_threshold \* 3 # canny边缘检测高阈值   alpha = 0.8 # 原图像权重  beta = 1. # 车道线图像权重  lambda\_ = 0.   imshape = image.shape # 获取图像大小   # 灰度图转换  gray = grayscale(image)   # 高斯滤波  blur\_gray = gaussian\_blur(gray, kernel\_size)   # Canny边缘检测  edge\_image = canny(blur\_gray, canny\_low\_threshold, canny\_high\_threshold)   # 生成Mask掩模  vertices = np.array([[(0, imshape[0]), (9 \* imshape[1] / 20, 11 \* imshape[0] / 18),  (11 \* imshape[1] / 20, 11 \* imshape[0] / 18), (imshape[1], imshape[0])]], dtype=np.int32)  masked\_edges = region\_of\_interest(edge\_image, vertices)   # 基于霍夫变换的直线检测  lines = hough\_lines(masked\_edges, rho, theta, hof\_threshold, min\_line\_len, max\_line\_gap)  line\_image = np.zeros\_like(image)   # 绘制车道线线段  draw\_lines(line\_image, lines, thickness=10)   # 图像融合  lines\_edges = weighted\_image(image, line\_image, alpha, beta, lambda\_)  return lines\_edges   def img\_detect\_demo():  img = cv2.imread('lane.jpg')  out = lane\_detect(img)  cv2.imshow("image", out)  cv2.waitKey()   def video\_detect\_demo():  cap = cv2.VideoCapture("project\_video.mp4")  count = 0  while(cap.isOpened()):  ret\_value, frame = cap.read()  if ret\_value == 0:  break  processed = lane\_detect(frame)  cv2.imshow("image", processed)  count += 1  print(count)  if cv2.waitKey(10) & 0xFF == ord('q'):  break  cap.release()  cv2.destroyAllWindows()   if \_\_name\_\_ == '\_\_main\_\_':  img\_detect\_demo()  # video\_detect\_demo() |

实验效果





代码分析及现象

Hough变换可以用于将边缘像素连接起来得到边界曲线，优点在于受噪声和曲线间断的影响较小，在已知曲线形状的条件下，Hough变换实际上是利用分散的边缘点进行曲线逼近，它也可看成是一种聚类分析技术

# 实验心得

在计算机视觉中，经常需要将一些特定的形状图形从图片中提取出来，如果直接用像素点来搜寻非常困难，这时候需要将图像从像素按照一定的算法映射到参数空间。hough变化提供了一种从图像像素信息到参数空间的变换方法。对于像直线，圆，椭圆这样的规则曲线hough是一种常用的算法。hough变化最大的优点在于特征边缘描述中间隔的容忍性并且该变换不受图像噪声的影响。