

## 1. 实验目的

学习 Python 的 OpenCV 库，了解图像处理的原理和基本操作；

学习并使用图像处理的算法（如大津算法）解决实际问题；

实现图像目标检测和轨迹跟踪，对不同算法的效果和适用范围进行对比分析。

## 2. 实验环境

Anaconda;

OpenCV;

Visual Studio Code.

### 3. 实验内容

#### 3.1 (对应实验二)

##### 3.1.1 实验要求:

- (1) 在开发环境下，自行选择有代表性的图像；
- (2) 显示对应的 B、G、R 通道；
- (3) 将原有图像转化为 HSV 空间表达，并显示对应分量；
- (4) 分别对 RGB 和 HSV 分量显示图像进行分析；
- (5) (可选)使用 matplotlib 的 pyplot，并应用 plot 函数显示图像，看是否能正常显示进一步使用该函数将图像分成 2\*4 个子窗口，分别在不同子窗口中显示 原始图像及不同分量

##### 3.1.2 实验设计

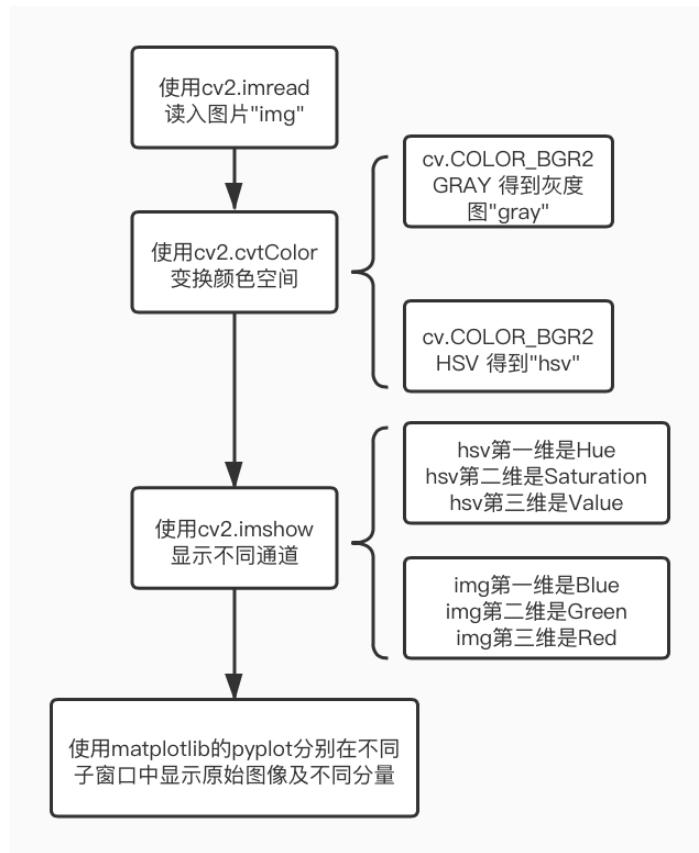


图 1. 实验二设计流程图

##### 3.1.3 算法实现

```
1. import cv2 as cv
2. import matplotlib.pyplot as plt
3.
4. filename = r'./Users/loujieming/Downloads/season.JPG'
5. img = cv.imread(filename)
6.
7. #转化颜色空间
8. gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
9. hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
10.
11. # 绘制子图, 由于 cv 读入图片信息为 BGR 信息, 使用 plt 时需要转换
12. plt.figure()
13. plt.subplot(2,4,1)
14. plt.imshow(img[:, :, [2, 1, 0]]) # 转化成 RGB
15. plt.title("img")
16. plt.subplot(2,4,2)
17. plt.imshow(img[:, :, [2, 1, 0]][:, :, 0])
18. plt.title("Blue")
19. plt.subplot(2,4,3)
20. plt.imshow(img[:, :, [2, 1, 0]][:, :, 1])
21. plt.title("Green")
22. plt.subplot(2,4,4)
23. plt.imshow(img[:, :, [2, 1, 0]][:, :, 2])
24. plt.title("Red")
25.
26. plt.subplot(2,4,5)
27. plt.imshow(hsv)
28. plt.title("hsv")
29. plt.subplot(2,4,6)
30. plt.imshow(hsv[:, :, [2, 1, 0]][:, :, 0])
31. plt.title("Hue")
32. plt.subplot(2,4,7)
33. plt.imshow(hsv[:, :, [2, 1, 0]][:, :, 1])
34. plt.title("Saturation")
35. plt.subplot(2,4,8)
36. plt.imshow(hsv[:, :, [2, 1, 0]][:, :, 2])
37. plt.title("Value")
38.
39. plt.show()
40.
41. # 使用 cv2 显示图片
42. cv.imshow('source image', img)
43. cv.imshow('gray', gray)
44. cv.waitKey()
45.
46. cv.imshow("Hue", hsv[:, :, 0])
```

```
47. cv.imshow("Saturation", hsv[:, :, 1])
48. cv.imshow("Value", hsv[:, :, 2])
49. cv.waitKey()
50.
51. cv.imshow("Blue", img[:, :, 0])
52. cv.imshow("Green", img[:, :, 1])
53. cv.imshow("Red", img[:, :, 2])
54.
55. cv.waitKey()
56. cv.destroyAllWindows()
```

### 3.1.4 结果分析

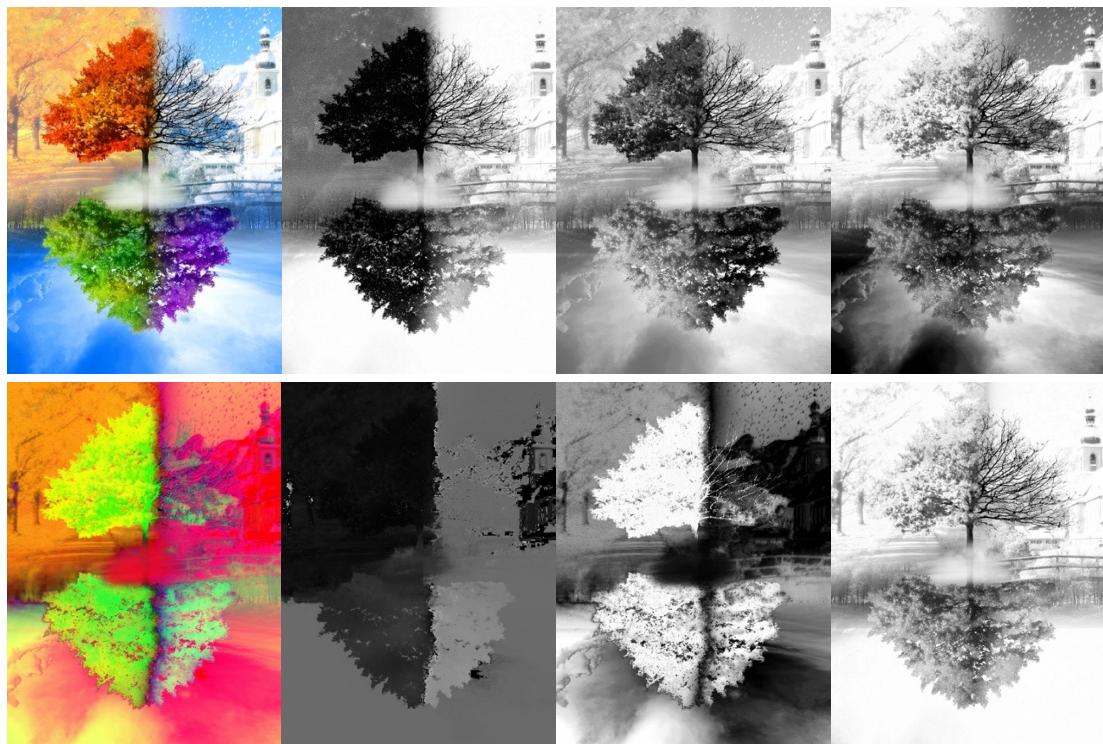


图 2. 使用 cv.imshow 作图

第一行从左到右为原图/Blue/Green/Red 通道， 第二行从左到右为 hsv 图像 /Hue/Saturation/Value 通道

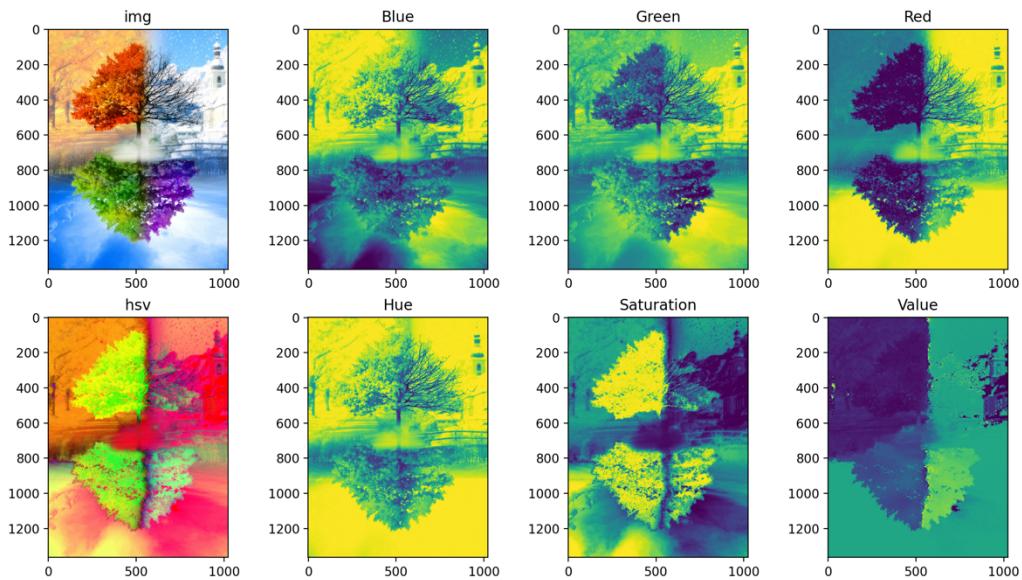


图 3. 使用 plt.imshow 绘图

这个实验主要用到了 RGB 和 HSV 两个颜色空间。

RGB 空间由三个通道表示一幅图像，分别为红色(R)，绿色(G)和蓝色(B)。这三种颜色的不同组合可以形成几乎所有的其他颜色，这种表示方法很适合图像的显示。直观地看，在绘制的 RGB 单通道图像中，深浅与这个像素的颜色值相关。

相比于 RGB 空间，HSV 颜色空间更接近人们对彩色的感知经验。非常直观地表达颜色的色调、鲜艳程度和明暗程度，方便进行颜色的对比。直观地看，在绘制的 HSV 单通道图像中，深浅与这个像素的色调、饱和度、明度相关。

对比可以看出cv和matplotlib在处理和显示图片有很多区别：

1. 读入时，`cv.imread`读入是BGR模式，而`plt.imread`读入是RGB模式，两者的通道顺序不同。使用plt和cv绘制图像时，经过通道顺序的对换，可以看出绘制的图像完全一致。
2. 当显示单独通道的图像时，`cv.imshow`默认为黑白图，而`plt.imshow`默认为黄绿图。

## 3.2 (对应实验四)

### 3.2.1 实验要求

- (1) 在开发环境下，显示米粒图像的灰度直方图；
- (2) 使用大津或其它方法进行阈值分割，得到分割后的二值化结果；
- (3) 对结果应用 `findContours` 函数，得到所有米粒对应的轮廓；
- (4) 画出每一米粒对应的最小包围矩形，进一步计算方差并进行统计；
- (5) 对分割及统计结果进行分析。

### 3.2.2 实验设计

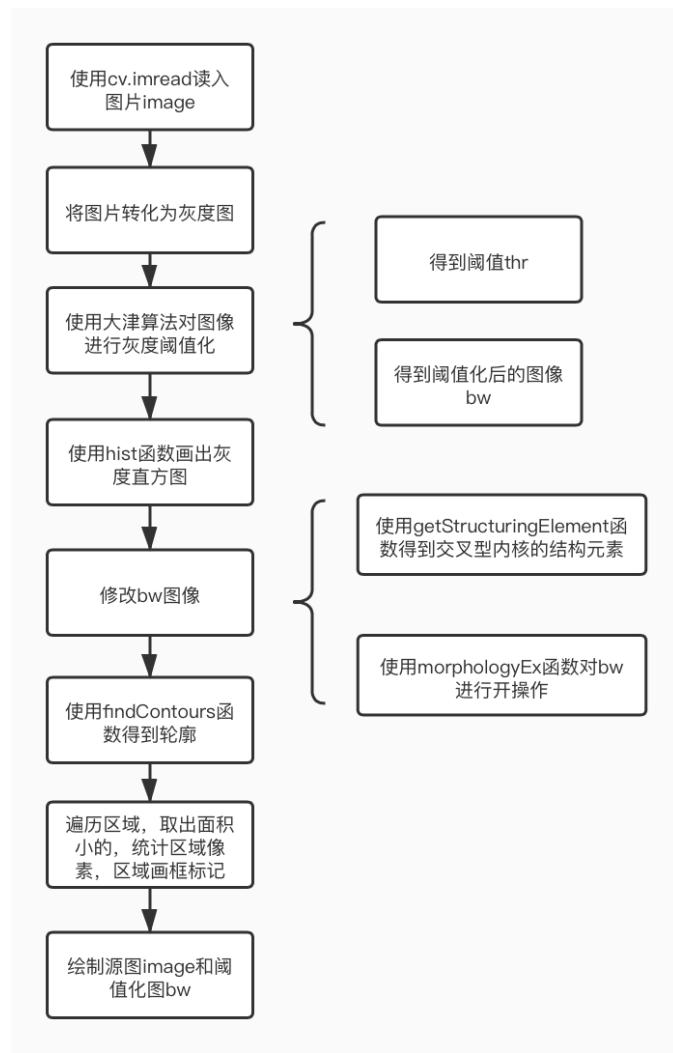


图4. 实验4设计流程图

### 3.2.3 算法实现

```
1 import cv2 as cv
2 import copy
3 import matplotlib.pyplot as plt
4 import numpy as np
5
```

```
6 def contrast_brightness_image(src1, a, g):
7     h, w, ch = src1.shape#获取 shape 的数值, height 和 width、通道
8
9     #新建全零图片数组 src2,将 height 和 width, 类型设置为原图片的通道类型(色素全为零, 输出为全黑图片)
10    src2 = np.zeros([h, w, ch], src1.dtype)
11    dst = cv.addWeighted(src1, a, src2, 1-a, g)#addWeighted 函数说明如下
12    cv.imshow("con-bri-demo", dst)
13    return dst
14
15 # 打开图像
16 filename = r'./Users/loujieming/Downloads/rice.JPG'
17 image = cv.imread(filename)
18 gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
19
20 # 修改图像对比度
21
22 # 大津算法灰度阈值化
23 # thr, bw = cv.threshold(gray,129,200,cv.THRESH_BINARY)
24 thr, bw = cv.threshold(gray, 0, 0xff, cv.THRESH_OTSU)
25 print('Threshold is :', thr)
26
27 # 画出灰度直方图
28 plt.hist(gray.ravel(), 256, [0, 256])
29 plt.show()
30
31 element = cv.getStructuringElement(cv.MORPH_CROSS, (3, 3))
32 bw = cv.morphologyEx(bw, cv.MORPH_OPEN, element)
33
34 seg = copy.deepcopy(bw)
35 # 计算轮廓
36 cnts, hier = cv.findContours(seg, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
37 minarea = 5000
38 maxarea = 0
39 meanarea = 0
40 count = 0
41 # 遍历所有区域, 并去除面积过小的
42 for i in range(len(cnts), 0, -1):
43     c = cnts[i-1]
44     area = cv.contourArea(c)
45     maxarea = max(maxarea,area)
46     minarea = min(minarea,area)
47     meanarea += area
48     if area < 10:
49         continue
50     count = count + 1
51     print("blob", i, " : ", area)
```

```
52
53     # 区域画框并标记
54     x, y, w, h = cv.boundingRect(c)
55     cv.rectangle(image, (x, y), (x+w, y+h), (0, 0, 0xff), 1)
56     cv.putText(image, str(count), (x, y), cv.FONT_HERSHEY_PLAIN, 0.5, (0, 0xff, 0))
57
58 meanarea /= len(cnts)
59 print("米粒数量: ", count)
60 print("maxarea=" + str(maxarea))
61 print("minarea=" + str(minarea))
62 print("meanarea=" + str(meanarea))
63
64 cv.startWindowThread()
65
66 cv.imshow("源图", image)
67 cv.imwrite("源图.jpg", image)
68 cv.waitKey()
69 cv.imshow("阈值化图", bw)
70 cv.imwrite("阈值化图.jpg", bw)
71 cv.waitKey()
72 cv.destroyAllWindows()
```

### 3.2.4 结果分析

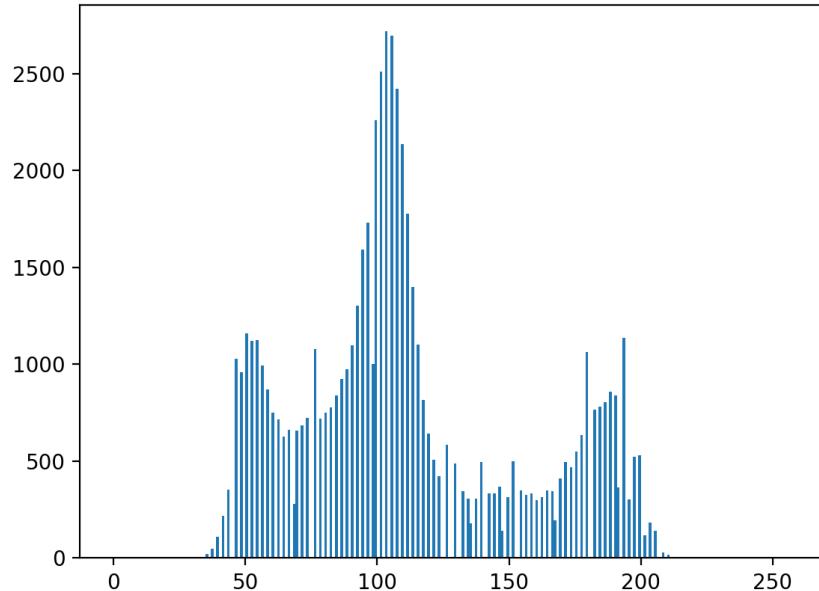


图5. 米粒图像的灰度直方图

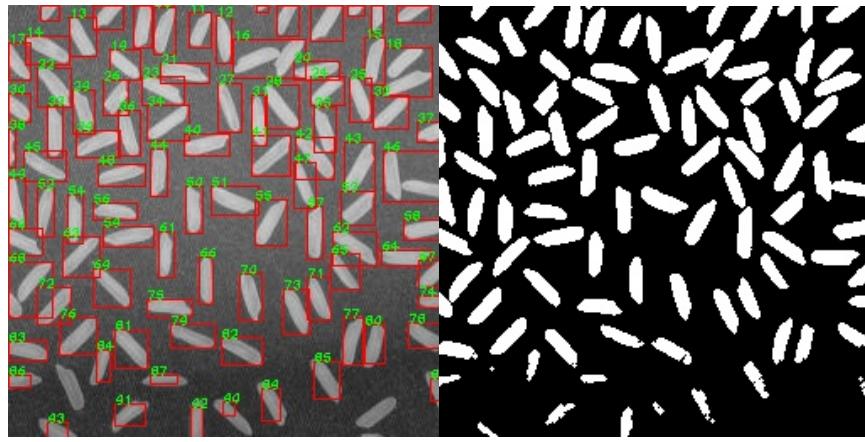


图6. 左图为源图，右图为阈值化图

对米粒结果的统计：

Max area	383.5
Min area	2.0
Mean area	146.397
Threshold	129.0
米粒数量	93

经过筛选的区域分割像素统计的详细结果见附录A。

观察灰度直方图发现，目标区和背景区之间或者不同目标区之间，灰度存在明显差异。观察图像可知，大津算法按图像的灰度特性，将图像分成背景和前景两部分。在源图上，将背景部分用闭合的红线框上并依次标号，总计 93 个米粒，正确率很高。阈值化图中也将背景清晰地标志出来了。进一步计算方差为 35.67。落在  $2.5\sigma$  范围内的米粒数量为 30 个。

然而，发现米粒图上仍然有一些米粒没有被标记出来。尝试通过调整米粒图像的对比度和亮度，仍然无法提高标记成功率；使用 `cv.THRESH_BINARY` 的方法调整阈值，仍然发现 129 的阈值是最佳的结果。这是因为 OpenCV 的大津算法是基于全局的阈值分割算法，极易受到光照不均的干扰，使图像前景像素与背景像素分割不合理，针对米粒图无法区分出较暗的米粒。更好的方法是实现局部的大津算法。但是 OpenCV 没有封装好的局部大津算法的函数。下面是局部大津算法的一种 Python 实现，主要思路是采用局部阈值分割的方法，以每列为局部，将每列进行一次大津算法，求出列阈值，以此类推，直至遍历完所有的列。代码实现如下：

```

1. import cv2 as cv
2. import numpy as np

3. # 转灰
4. def rgb2gray(img):
5.     h=img.shape[0]
6.     w=img.shape[1]
7.     img1=np.zeros((h,w),np.uint8)
8.     for i in range(h):
9.         for j in range(w):
10.             img1[i,j]=0.144*img[i,j,0]+0.587*img[i,j,1]+0.299*img[i,j,1]
```

```

11.     return img
12.
13. # 局部大津算法实现
14. def otsu(img):
15.     h=img.shape[0]
16.     w=img.shape[1]
17.     otsuimg=np.zeros((h,w),np.uint8)
18.     for i in range(w):  # 遍历列
19.         sigma=threshold=0  # 定义类间方差和最终阈值
20.         histogram=np.zeros(256,np.int32)  # 初始化各灰度级个数统计
21.         probability=np.zeros(256,np.float32)  # 初始化各灰度级概率分布
22.         for j in range (h):  # 遍历行, 进行 otsu 算法
23.             s=img[j,i]
24.             histogram[s]+=1  # 统计灰度级中每个像素在整幅图像中的个数
25.             for k in range (256):
26.                 probability[k]=histogram[k]/h  # 统计每个灰度级占图像中的分布
27.             for p in range (255):
28.                 w0 = w1 = 0  # 定义前景像素点和背景像素点灰度级占图像中的分布
29.                 fgs = bgs = 0  # 定义前景像素点灰度级总和 and 背景像素点灰度级总和
30.                 for q in range (256):
31.                     if q<=p:  # 当前 i 为分割阈值
32.                         w0+=probability[q]  # 前景像素点占整幅图像的比例累加
33.                         fgs+=q*probability[q]  # 前景像素点的平均灰度
34.                     else:
35.                         w1+=probability[q]  # 背景像素点占整幅图像的比例累加
36.                         bgs+=q*probability[q]  # 背景像素点的平均灰度
37.                     u0=fgs/w0
38.                     u1=bgs/w1
39.                     g=w0*w1*(u0-u1)**2  # 类间方差
40.                     if g>=sigma:
41.                         sigma=g
42.                         threshold=p
43.                     for j in range (h):  # 对某列的每一行进行二值化
44.                         if img[j,i]>threshold:
45.                             otsuimg[j,i]=255
46.                         else:
47.                             otsuimg[j,i]=0
48.     return otsuimg

```

局部大津法得到的边缘结果为:



图 7. 左图为全局大津法得到的阈值化图，右图为局部大津法得到的阈值化图  
可以看出，有些亮度较低的米粒也被识别出来了。如果想实现更好的识别度，需要进行更细  
粒度的划分并在小区域上进行大津法。

### 3.3 (对应实验五，重点)

#### 3.3.1 实验要求

- (1) 系统输入:给定视频(含有相关目标); 系统输出:检测的目标框及目标运动轨迹;
- (2) 首先在“viplane”视频上进行实验;进一步在“Cap02t3”、“999”和 “video1”视频上进行实验。

提示:

- (1) 运动目标检测可利用 OpenCV 提供的背景提取算法;
- (2) 运动目标跟踪可利用 OpenCV 提供的多目标跟踪方法, 如 KCF 等;
- (3) (可选)为得到更好效果, 可尝试利用深度学习进行目标检测;
- (4) (可选)为得到更好的多目标跟踪效果, 可尝试利用 SORT、SiamRPN 等方法。

#### 3.3.2 实验设计

##### 3.3.2.1 运动目标检测

使用了背景提取算法, 进行将一幅图作为背景, 然后和每一帧对比。

##### 3.3.2.2 多目标跟踪

###### (1) 使用光流法实现多目标跟踪 (自动选择目标跟踪)

光流法是进行视频中运动对象轨迹标记的一种很常用的方法, 在 OpenCV 中实现光流也很容易。光流表达了图像的变化, 由于它包含了目标运动的信息, 因此可被观察者用来确定目标的运动情况。`cv2.calcOpticalFlowPyrLK` 函数计算一个稀疏特征集的光流, 使用金字塔中的迭代 Lucas-Kanade 方法。

实现原理:

首先选取第一帧, 在第一帧图像中检测 Shi-Tomasi 角点, 然后使用 LK 算法来迭代的跟踪这些特征点。迭代的方式就是不断向 `cv2.calcOpticalFlowPyrLK` 中传入上一帧图片的特征点以及当前帧的图片。函数会返回当前帧的点, 这些点带有状态 1 或者 0, 如果在当前帧找到了上一帧中的点, 那么这个点的状态就是 1, 否则就是 0。

实现流程:

1. 加载视频;
2. 调用 `GoodFeaturesToTrack` 函数寻找兴趣点 (关键点) ;
3. 调用 `CalcOpticalFlowPyrLK` 函数计算出两帧图像中兴趣点的移动情况;
4. 删除未移动的兴趣点;
5. 在两次移动的点之间绘制一条线段。

###### (2) 使用KCF算法实现多目标跟踪 (可以跟踪指定的多目标)

Kernel Correlation Filter (KCF), 即核相关滤波算法。KCF 算法使用目标周围区域的循环矩阵采集正负样本, 利用脊回归训练目标检测器, 并成功的利用循环矩阵在傅里叶空间可对

角化的性质将矩阵的运算转化为向量的 Hadamad 积，即元素的点乘，大大降低了运算量，提高了运算速度，使算法满足实时性要求。将线性空间的脊回归通过核函数映射到非线性空间，在非线性空间通过求解一个对偶问题和某些常见的约束，同样的可以使用循环矩阵傅里叶空间对角化简化计算。给出了一种将多通道数据融入该算法的途径。

直接使用 OpenCV 库中的 `TrackerKCF_create()` 函数创建 KCF 算法跟踪器，即可实现。

### 3.3.3 算法实现

#### 3.3.3.1 运动目标检测

```
1. import cv2
2. import numpy as np

3. cap = cv2.VideoCapture('/Users/loujieming/小铭不熬夜/作业和笔记/图像工程/图片视频/999.mp4')

4. # 测试用，查看视频 size
5. size = (int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)),
6.           int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT)))
7. print('size: '+repr(size))

8.
9. # 构建椭圆结果
10. es = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (9, 4))
11. kernel = np.ones((5, 5), np.uint8)
12. background = None
13.
14. while True:
15.     # 读取视频流
16.     grabbed, frame = cap.read()
17.
18.     if frame is None:
19.         Break
20.
21.     # 对帧进行预处理，>>转灰度图>>高斯滤波（降噪：摄像头震动、光照变化）。
22.     gray_lwpCV = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
23.     gray_lwpCV = cv2.GaussianBlur(gray_lwpCV, (21, 21), 0)
24.
25.     if background is None:
26.         background = gray_lwpCV
27.         Continue
28.
29.     # 对比背景之后的帧与背景之间的差异，并得到一个差分图 (different map)。
30.     # 阈值（二值化处理）>>膨胀 (dilate) 得到图像区域块
31.     diff = cv2.absdiff(background, gray_lwpCV)
32.     diff = cv2.threshold(diff, 25, 255, cv2.THRESH_BINARY)[1]
33.     diff = cv2.dilate(diff, es, iterations=2)
34.
35.     _, fgmask = cv2.threshold(diff.copy(), 0, 0xff, cv2.THRESH_BINARY)
36.
```

```

37.     # 显示矩形框：计算一幅图像中目标的轮廓
38.     cnts, hier = cv2.findContours(fgmask.copy(), cv2.RETR_EXTERNAL,
39.                                   cv2.CHAIN_APPROX_SIMPLE)
40.
41.     # 遍历所有区域，并去除面积过小的
42.     for i in range(len(cnts), 0, -1):
43.         c = cnts[i-1]
44.         area = cv2.contourArea(c)
45.         if area < 600:
46.             Continue
47.
48.     # 区域画框并标记
49.     x, y, w, h = cv2.boundingRect(c)
50.     frame = cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
51.
52.     cv2.imshow('contours', frame)
53.     # cv2.imshow('dis', diff)
54.
55.     key = cv2.waitKey(100) & 0xFF
56.     if key == ord('q'):    # 按'q'键退出循环
57.         break
58. # 释放资源并关闭窗口
59. cap.release()
60. cv2.destroyAllWindows()

```

### 3. 3. 3. 2 多目标跟踪

#### (1) 使用光流法实现多目标跟踪

```

1. import numpy as np
2. import cv2 as cv
3.
4. cap = cv.VideoCapture('/Users/loujieming/小铭不熬夜/作业和笔记/图像工程/图片视频
5. /viplane.avi')
6.
7. # ShiTomasi corner detection 的参数
8. feature_params = dict(maxCorners=100,
9.                         qualityLevel=0.3,
10.                        minDistance=7,
11.                        blockSize=7)
12. # 光流法参数
13. # maxLevel 未使用的图像金字塔层数
14. lk_params = dict(winSize=(15, 15),
15.                   maxLevel=2,
16.                   criteria=(cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 0.03))
17.
18. # 创建随机生成的颜色
19. color = np.random.randint(0, 255, (100, 3))

```

```

20.
21. ret, old_frame = cap.read() # 取出视频的第一帧
22. old_gray = cv.cvtColor(old_frame, cv.COLOR_BGR2GRAY) # 灰度化
23. p0 = cv.goodFeaturesToTrack(old_gray, mask=None, **feature_params) # 获得旧的关注点
24. mask = np.zeros_like(old_frame) # 为绘制创建掩码图
25.
26. while True:
27.     _, frame = cap.read()
28.     frame_gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
29.
30.     # 计算光流以获取点的新位置
31.     p1, st, err = cv.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None,
32.                                            **lk_params)
33.     # 选择 good points
34.     good_new = p1[st == 1]
35.     good_old = p0[st == 1]
36.     # 绘制跟踪框
37.     for i, (new, old) in enumerate(zip(good_new, good_old)):
38.         a, b = new.ravel()
39.         c, d = old.ravel()
40.         mask = cv.line(mask, (int(a), int(b)), (int(c), int(d)), color[i].tolist(),
41.                      1)
42.         frame = cv.circle(frame, (int(a), int(b)), 10, color[i].tolist())
43.     img = cv.add(frame, mask)
44.     cv.imshow('frame', img)
45.     k = cv.waitKey(100) # & 0xff
46.     if k == 27:
47.         Break
48.     old_gray = frame_gray.copy()
49.     p0 = good_new.reshape(-1, 1, 2)
50.
51. cv.destroyAllWindows()
52. cap.release()

```

## (2) 使用KCF算法实现多目标跟踪

```

1. import cv2
2. import sys
3.
4. filename = '/Users/loujieming/小铭不熬夜/作业和笔记/图像工程/图片视频/Cap02t3.avi'
5. video = cv2.VideoCapture(filename)
6.
7. ok, frame = video.read()
8. if not ok:
9.     print('Cannot read video file')

```

```
10.     sys.exit()
11.
12. bbox = (287, 23, 86, 320)
13.
14. bbox1 = cv2.selectROI(frame, False)
15. bbox2 = cv2.selectROI(frame, False)
16. print(bbox1, bbox2)
17.
18. tracker1 = cv2.TrackerKCF_create()
19. tracker2 = cv2.TrackerKCF_create()
20. ok1 = tracker1.init(frame, bbox1)
21. ok2 = tracker2.init(frame, bbox2)
22.
23. p31 = (int(bbox1[0] + bbox1[2]/2), int(bbox1[1] + bbox1[3]/2))
24. p32 = (int(bbox2[0] + bbox2[2]/2), int(bbox2[1] + bbox2[3]/2))
25. list1 = [p31]
26. list2 = [p32]
27.
28. while True:
29.     # Read a new frame
30.     ok, frame = video.read()
31.
32.     # Start timer
33.     timer = cv2.getTickCount()
34.
35.     # Update tracker
36.     ok1, bbox1 = tracker1.update(frame)
37.     ok2, bbox2 = tracker2.update(frame)
38.     print(bbox1, bbox2)
39.
40.     # Calculate Frames per second (FPS)
41.     fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer)
42.
43.     # Draw bounding box
44.     if ok1:
45.         # Tracking success
46.         p1 = (int(bbox1[0]), int(bbox1[1]))
47.         p2 = (int(bbox1[0] + bbox1[2]), int(bbox1[1] + bbox1[3]))
48.         p0 = (int(bbox1[0] + bbox1[2]/2), int(bbox1[1] + bbox1[3]/2))
49.         cv2.rectangle(frame, p1, p2, (255, 0, 0), 2, 1)
50.         list1.append(p0)
51.         for i in range(len(list1)-1):
52.             mask1 = cv2.line(mask1, list1[i], list1[i+1], (0,255,0), 1)
53.     else:
54.         # Tracking failure
```

```

55.     cv2.putText(frame, "Tracking failure detected", (100, 80),
56.                   cv2.FONT_HERSHEY_SIMPLEX,
57.                   0.75, (0, 0, 255), 2)
58.
59.     if ok2:
60.         # Tracking success
61.         p1 = (int(bbox2[0]), int(bbox2[1]))
62.         p2 = (int(bbox2[0] + bbox2[2]), int(bbox2[1] + bbox2[3]))
63.         cv2.rectangle(frame, p1, p2, (255, 0, 0), 2, 1)
64.         p0 = (int(bbox1[0] + bbox1[2]/2), int(bbox1[1] + bbox1[3]/2))
65.         cv2.rectangle(frame, p1, p2, (255, 0, 0), 2, 1)
66.         list1.append(p0)
67.         for i in range(len(list1)-1):
68.             mask1 = cv2.line(mask1, list1[i], list1[i+1], (0,255,0), 1)
69.     else:
70.         # Tracking failure
71.         cv2.putText(frame, "Tracking failure detected", (100, 80),
72.                     cv2.FONT_HERSHEY_SIMPLEX,
73.                     0.75, (0, 0, 255), 2)
74.
75.         # Display FPS on frame
76.         # cv2.putText(frame, "FPS : " + str(int(fps)), (100, 50),
77.                     cv2.FONT_HERSHEY_SIMPLEX,
78.                     0.75, (50, 170, 50), 2)
79.
80.         # Display result
81.         cv2.imshow("Tracking", frame)
82.
83.         # Exit if ESC pressed
84.         k = cv2.waitKey(100) & 0xff
85.         if k == 27: break
86.         elif k == ord("q"):
87.             Break
88.
89. video.release()
90. cv2.destroyAllWindows()

```

### 3.3.4 结果分析

#### 3.3.4.1 运动目标检测

视频的截图结果为：

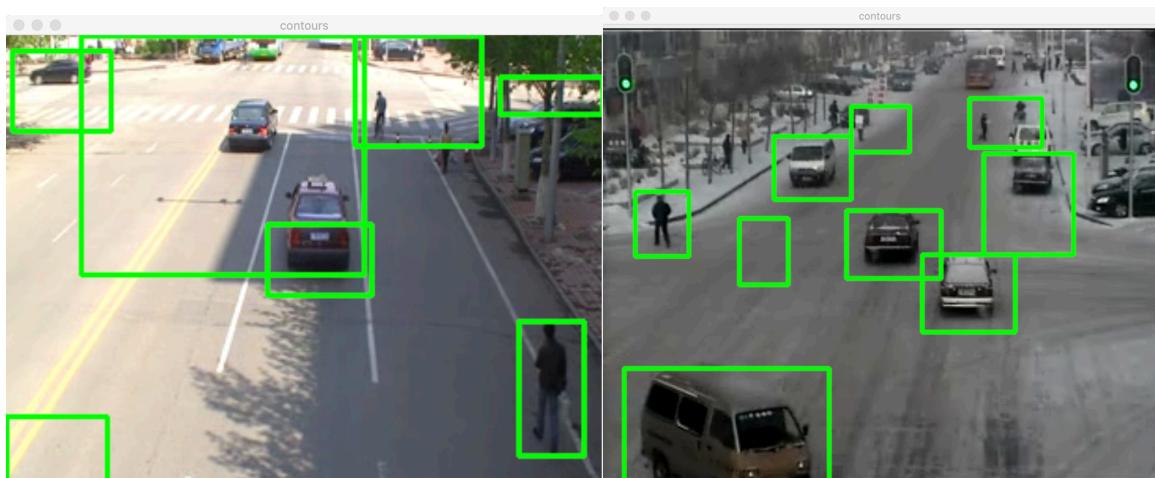
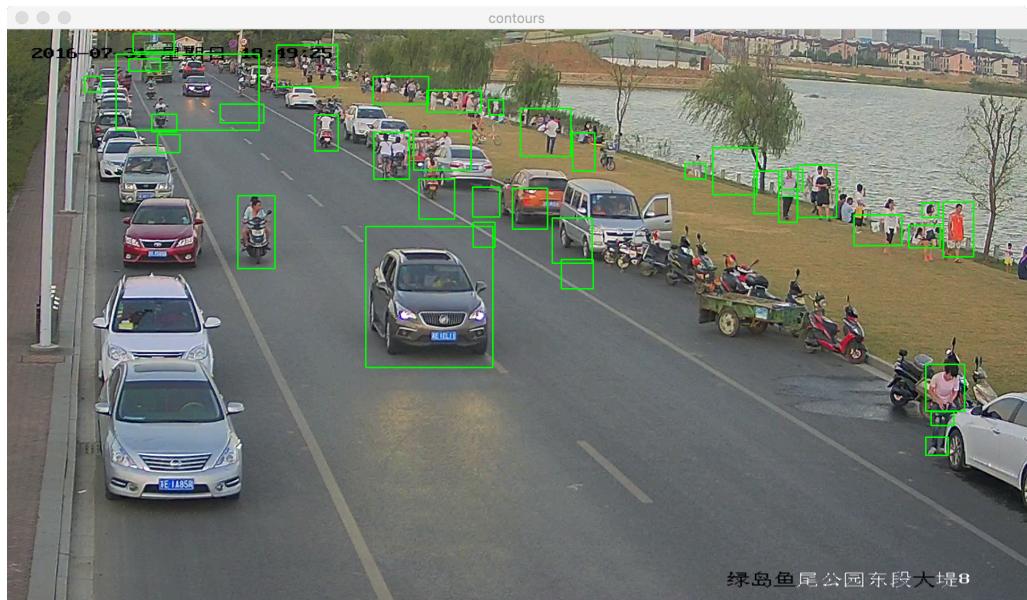


图8. 上：999.mp4，下左：video1.avi，下右：Cap02t3.avi

可以发现，采用背景提取算法的视频目标检测可以检测之前背景没有的景象。但是一开始存入的背景可能随光照变化而造成错误，可以用在光照环境稳定的地方。而且这种检测方法只能检测出帧之间的不同点，无法实现对特定目标的跟踪。

### 3.4.2 多目标跟踪

#### (1) 使用光流法实现多目标跟踪



图9. 上: 999.mp4, 下左: viplane.avi, 下右: Cap02t3.avi

可以发现，光流法将存在明显运动的特征点自动检测出来，并同时跟踪多个特征点的轨迹，效果很好。这种方法对视频中新出现的特征点也能做到很好的检测。但是，光流法无法对特定的目标进行跟踪，其检测的特征点也不一定具有实际意义。

## (2) 使用KCF算法实现多目标跟踪



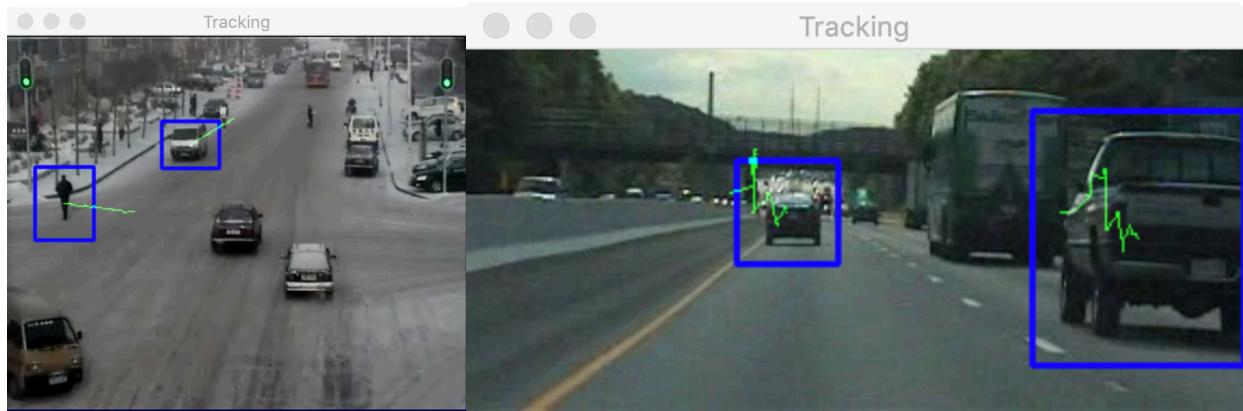


图10. 上: 999.mp4, 下左: Cap02t3.avi, 下右: viplane.avi

本文使用掩层和Opencv-contrib-python中自带的跟踪器，实现了两个特定目标的同时跟踪并绘制其移动轨迹。跟踪效果良好。如果想实现更多目标的跟踪，只需要在代码中增加更多的跟踪器即可。当目标离开画面时，视频中会显示"Tracking failure detected"字样，如图所示。



图11. video1.avi 目标离开画面

#### 4 实验结论

- (1) 实现了图像处理的基本操作，了解了颜色空间，分别显示对应的 B、G、R 通道以及将原有图像转化为 HSV 空间表达，并显示对应分量；对比了使用 matplotlib 的 pyplot 和 cv2 对图像显示的区别。
- (2) 使用并改进了全局大津算法，画出了米粒图的灰度直方图并进行分析，实现了对米粒图像的识别和统计。成功画出米粒对应的最小包围矩形。
- (3) 分别使用了背景提取算法、光流法和 KCF 算法对不同视频进行目标检测和目标轨迹跟踪。对效果和适用范围进行了对比分析，对算法应用进行一系列改进。

## 5 附录

### 附录A：区域像素大小

blob	97	96	95	93	92	91	90
像素数	49	113.5	166	12	104.5	166.5	36
blob	96	95	93	92	91	90	89
像素数	113.5	166	12	104.5	166.5	36	76
blob	95	93	92	91	90	89	88
像素数	166	12	104.5	166.5	36	76	101
blob	93	92	91	90	89	88	87
像素数	12	104.5	166.5	36	76	101	178
blob	92	91	90	89	88	87	86
像素数	104.5	166.5	36	76	101	178	123
blob	91	90	89	88	87	86	85
像素数	166.5	36	76	101	178	123	160
blob	90	89	88	87	86	85	84
像素数	36	76	101	178	123	160	168.5
blob	89	88	87	86	85	84	83
像素数	76	101	178	123	160	168.5	193.5
blob	88	87	86	85	84	83	82
像素数	101	178	123	160	168.5	193.5	168
blob	87	86	85	84	83	82	81
像素数	178	123	160	168.5	193.5	168	383.5
blob	86	85	84	83	82	81	80
像素数	123	160	168.5	193.5	168	383.5	122
blob	85	84	83	82	81	80	79
像素数	160	168.5	193.5	168	383.5	122	356.5
blob	84	83	82	81	80	79	78
像素数	168.5	193.5	168	383.5	122	356.5	143.5