Features

زهرا نیازی

اطلاعات گزارش

تاریخ: 13/09/1401

واژگان کلیدی:

چکیده

در این تمرین ابتدا چگونگی پیاده سازی الگوریتم Harris Feature Detection بیان شده و نتایج آن بررسی می شود. سپس روشی برای محاسبه تبدیل هندسی با استفاده از مچ کردن دو تصویر و سپس بازسازی تصویر اولیه توسط اعمال ماتریس تبدیل به تصویر معرفی شده است.

1-مقدمه

فیچر پوینت ها در تصویر نقاطی هستند که با تغییرات مختلف تصویر، تغییر نمی کنند و می توان باز هم آنها را شناسایی کرد. به این ترتیب در تصویر های مختلف از یک شی یا محیط، می توان با پیدا کردن این نقاط، تصاویر را برای کاربرد های مختلف تطبیق داد.

2-شرح تکنیکال

Harris Corner Detector 1-2

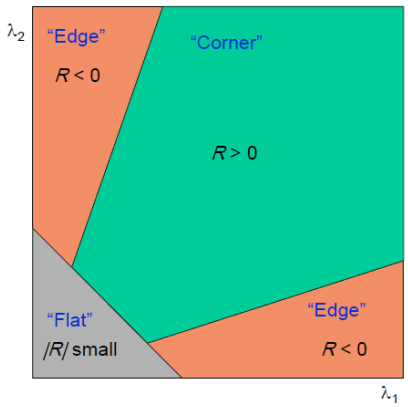
1-1-2

در تمرین اول از ما خواسته شده از آشکارساز Harris Corner را پیاده سازی کنیم، سپس با استفاده از آن نقاط ویژگی را استخراج کنیم. به این ترتیب آشکارساز Harris Corner را برای حداقل 4 مقیاس مختلف اعمال کنیم. سپس باید بررسی کنیم که کدام نقاط ویژگی را مشاهده می کنیم که در تمام این مقیاس های مختلف شناسایی شده اند.

نحوه عملکرد الگوریتم هریس به این شکل است که این به کمک مشتقات تصویر در جهت عمودی و افقی ماتریس ممان دوم را محاسبه کرده، سپس با محاسبه R توسط لاندا، و مقایسه نقاطی که در آن R از threshold خاصی بیشتر هستند، نقاط ویژگی را شناسایی میکند.

$$M = \sum w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

$$I_x \Leftrightarrow \frac{\partial I}{\partial x} \qquad I_y \Leftrightarrow \frac{\partial I}{\partial y} \qquad I_x I_y \Leftrightarrow \frac{\partial I}{\partial x}\frac{\partial I}{\partial y}$$



* zahra.niazi2001@gmail.com

درواقع مراحل الگوریتم هریس طبق زیر است:

1. مشتقات گاوسی را در هر پیکسل محاسبه کنید
2. ماتریس ممان دوم M را در یک پنجره گاوسی در اطراف هر پیکسل محاسبه کنید.
3. محاسبه تابع پاسخ گوشه R
4. آستانه گذاری روی R
5. ماکزیمم محلی تابع پاسخ را بیابید (non maximum supression)

2-2 Scene stitching with SIFT/SURF features

1-2-2

در این تمرین خواسته شده از پیاده سازی OpenCVو Python در عملگر SIFT برای یافتن نقاط ویژگی و ایجاد ارتباط بین تصاویر استفاده کنیم. در این حالت می توان مستقیماً بردارهای ویژگی نقاط مورد علاقه را با هم مقایسه کرد.

3-نتایج

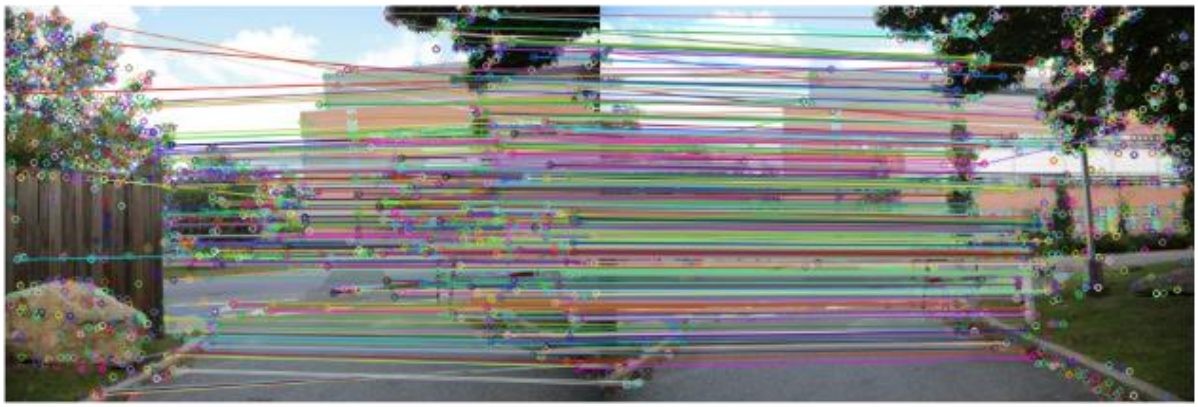
1-3

1-1-3

در scale های مختلف، ابتدا تصویر را resize کرده سپس با کمک تابع پیاده سازی شده برای الگوریتم هریس نقاط ویژگی را پیدا میکنیم.

با افزایش سایز تصویر ورودی گوشه های بزرگتری شناسایی می شوند و مقدار threshold باید کمتر باشد تا نقاط بیشتری شناسایی شوند اما بدون تغییر آن، تعداد نقاط شناسایی شده کاهش می یابد و همانطور که در تصویر مشخص است با افزایش سایز تصویر یعنی scale بالاتر، تعداد نقاط ویژگی شناسایی شده کاهش می یابد چرا که مقدار threshold را ثابت گرفتیم.

2-3
1-2-3
در این تمرین از ما خواسته شده با استفاده از الگوریتم sift و surf فرایند matching را انجام دهیم و فیچر پوینت های تصاویر را با یکدیگر تطبیق دهیم. در این الگوریتم ها برای matching از فاصله ی اقلیدسی جفت نقاط استفاده می شود.



4-Code

https://colab.research.google.com/drive/1Pbf-2qCNVCN_Iu__rb_ZntJXc_rsZa6Y?usp=sharing

```
def harris_corners(img, window_size):
    smooth = cv2.GaussianBlur(img,(3,3),0)
    gray = cv2.cvtColor(smooth, cv2.COLOR_BGR2GRAY)
    grad_x =  cv2.Sobel(gray,-1,1,0,ksize = 3)
    grad_y =  cv2.Sobel(gray,-1,0,1,ksize = 3)

    h,w = grad_x.shape
    corners = np.zeros((h,w), dtype = np.float64)
    grad_x = grad_x.astype(np.float64)
    grad_y = grad_y.astype(np.float64)

    grad_xx = grad_x**2
    grad_xx = cv2.GaussianBlur(grad_xx,(3,3),0.5)
    grad_yy = grad_y**2
    grad_yy = cv2.GaussianBlur(grad_yy,(3,3),0.5)
    grad_xy = grad_x *grad_y
    grad_xy = cv2.GaussianBlur(grad_xy, (3,3), 0.5)



    offset = int(window_size/2)
    for i in range(offset, h-offset):
        for j in range(offset, w-offset):
            temp1 = grad_xx[i-offset:i+offset+1, j-offset:j+offset+1]
            temp2 = grad_yy[i-offset:i+offset+1, j-offset:j+offset+1]
            temp3 = grad_xy[i-offset:i+offset+1, j-offset:j+offset+1]
            a = np.sum(temp1)
            b = np.sum(temp2)
            c= np.sum(temp3)
            det  = a*b - (c*c)
            trace = a+b
            R = det - 0.04*(trace**2)
            corners[i,j] = R

    return corners
```

```
def draw_corners(corner, img, scale, marker_size):
  y, x = np.nonzero(corner > 0.4 * np.max(corner))
  for i in range(len(x)):
      x1 = int(x[i]); y1 = int(y[i])
      cv2.circle(img, (x1,y1), marker_size ,(0,0,255),-1)

  imshow(img,title=[f'scale = {scale}'], figsize=10)



def corner_detector(image, i, w_size):
  img1 = image.copy()
  x , y ,channal= img1.shape

  img  = cv2.resize(img1, (int(y * i),int(x * i)))
  c = harris_corners(img, w_size)
  draw_corners(c, img, i, int(i*3))



corner_detector(harris, 1, 5)
corner_detector(harris, 2, 5)
corner_detector(harris, 0.5, 3)
corner_detector(harris, 0.8, 3)




def sift(img1, img2, img3):
  sift = cv2.xfeatures2d.SIFT_create()
  keypoints1, descriptors1 = sift.detectAndCompute(img1, None)
  keypoints2, descriptors2 = sift.detectAndCompute(img2, None)
  keypoints3, descriptors3 = sift.detectAndCompute(img3, None)

  bf = cv2.BFMatcher()
  matches12 = bf.knnMatch(descriptors1, descriptors2, k=2)
  matches13 = bf.knnMatch(descriptors1, descriptors3, k=2)
  matches23 = bf.knnMatch(descriptors2, descriptors3, k=2)

  good_matches12 = []
```

```
  for m, n in matches12:
      if m.distance < 0.75 * n.distance:
          good_matches12.append(m)

  good_matches13 = []
  for m, n in matches13:
      if m.distance < 0.75 * n.distance:
          good_matches13.append(m)

  good_matches23 = []
  for m, n in matches23:
      if m.distance < 0.75 * n.distance:
          good_matches23.append(m)

  img_matches12 = cv2.drawMatches(img1, keypoints1, img2, keypoints2, go
od_matches12, None)
  img_matches13 = cv2.drawMatches(img1, keypoints1, img3, keypoints3, go
od_matches13, None)
  img_matches23 = cv2.drawMatches(img2, keypoints2, img3, keypoints3, go
od_matches23, None)

  return img_matches12, img_matches13, img_matches23




matches = sift(sl,sm,sr)
imshow(matches[0],matches[1], matches[2])
```