МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики.

Домашнее задание №3

Обнаружение ключевых точек на изображении в целях оценивания поля движения

Компьютерное зрение

Выполнил студент группы М3403:
Давлетов Артем Эдуардович

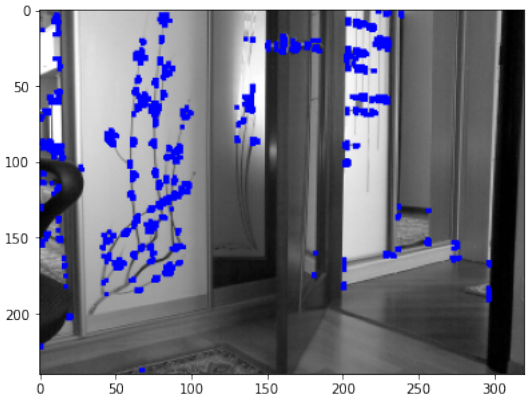
Преподаватель:
Титаренко Михаил Алексеевич

2020

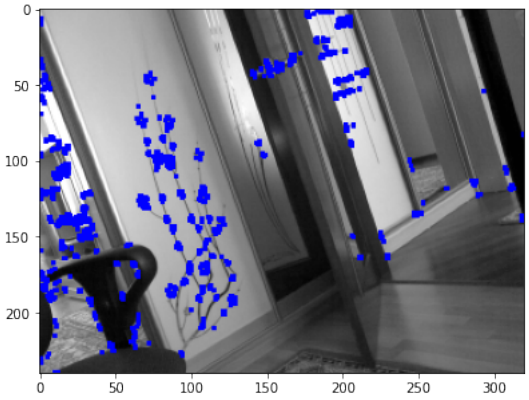
Вариант 4.

Цель работы: освоение способов обнаружения характерных точек на изображении, которые могут использоваться как для сопоставления двух изображений, так и для оценивания параметров движения камеры.

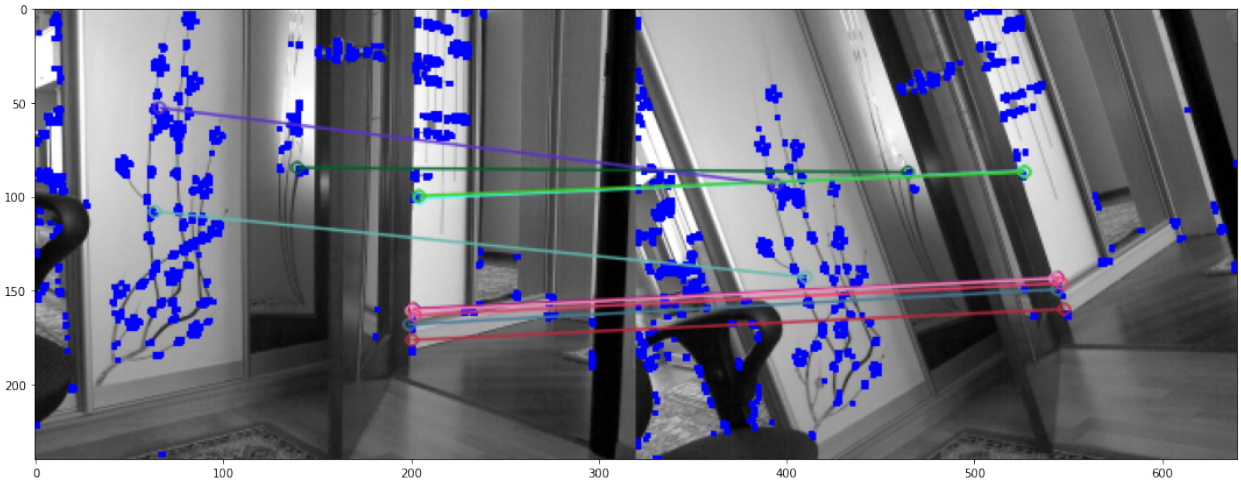
1. Для начала выделим ключевые точки на первом изображении с помощью функции `cv2.cornerHarris()`, которая в свою очередь находит углы(точки интереса) на изображении. Поскольку угол это пересечение двух ребер, он представляет собой точку, в которой направления этих двух ребер *меняются* . Следовательно, градиент изображения (в обоих направлениях) сильно варьируется, что может быть использовано для его обнаружения.



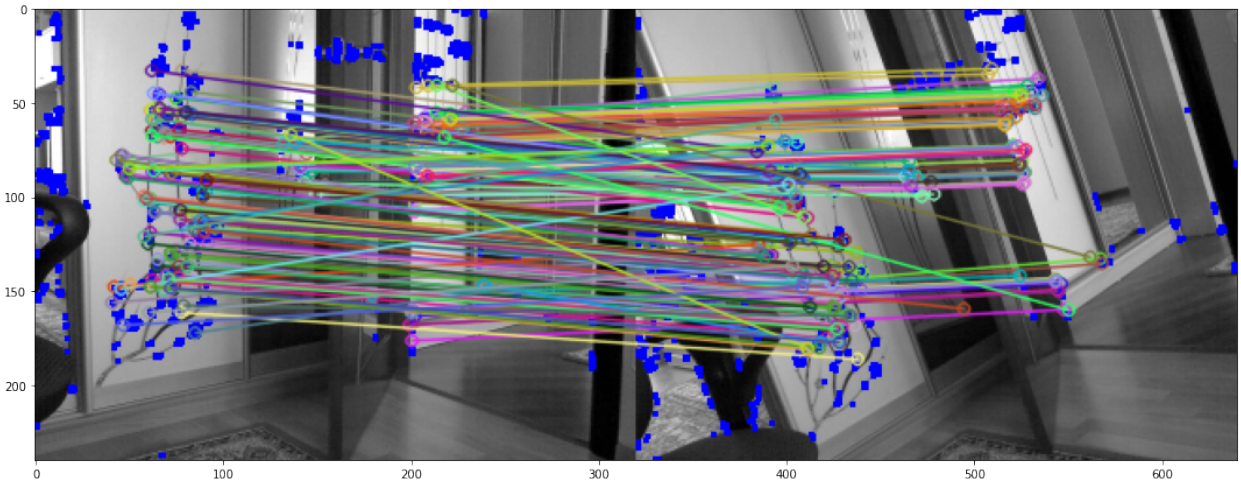
2. Теперь выделим ключевые точки на втором изображении с использованием той же функции.



3. Теперь, с помощью функции `cv2.BFMatcher()` найдем на изображениях 10 схожих точек и отобразим их:

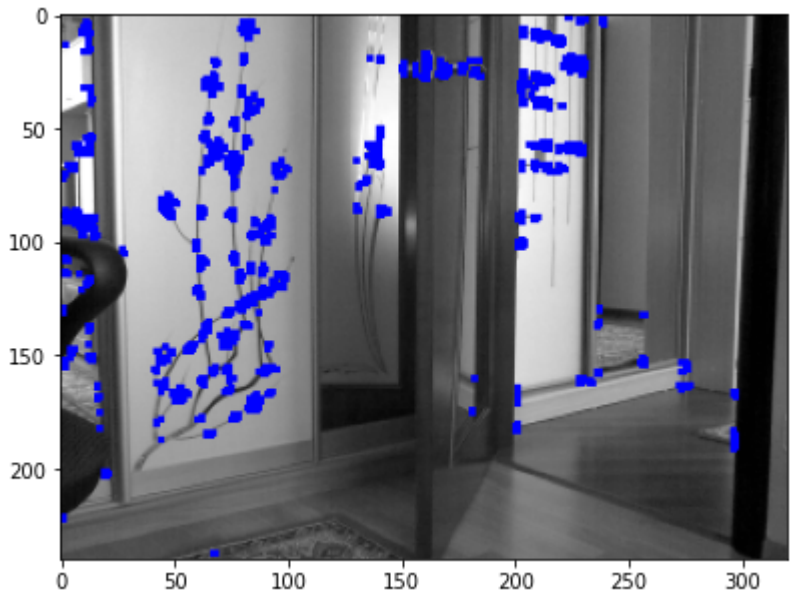


4. Попытаемся отобразить все совместные точки, но из-за большого их количества будет сложно определить их все:



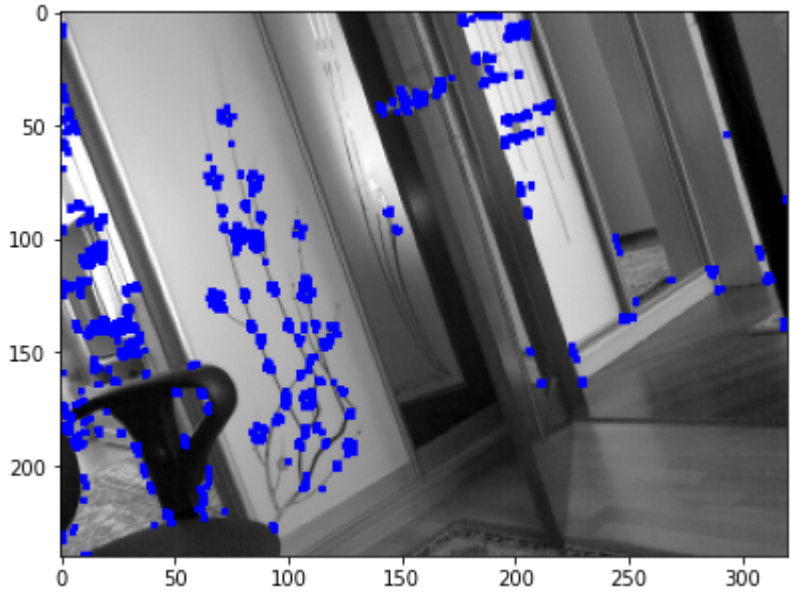
Реализация прикреплена ниже в виде jupyter notebook:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt


img1 = cv2.imread('img4_1.bmp')
gray = cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)
dst1 = cv.cornerHarris(gray,2,3,0.04)

# result is dilated for marking the corners, not important
dst1 = cv2.dilate(dst1,None)

# Threshold for an optimal value, it may vary depending on the image.
img1[dst1>0.01*dst1.max()]=[0,0,255]


plt.imshow(img1)
fig = plt.gcf()
fig.set_size_inches(10, 5)
plt.show()
```

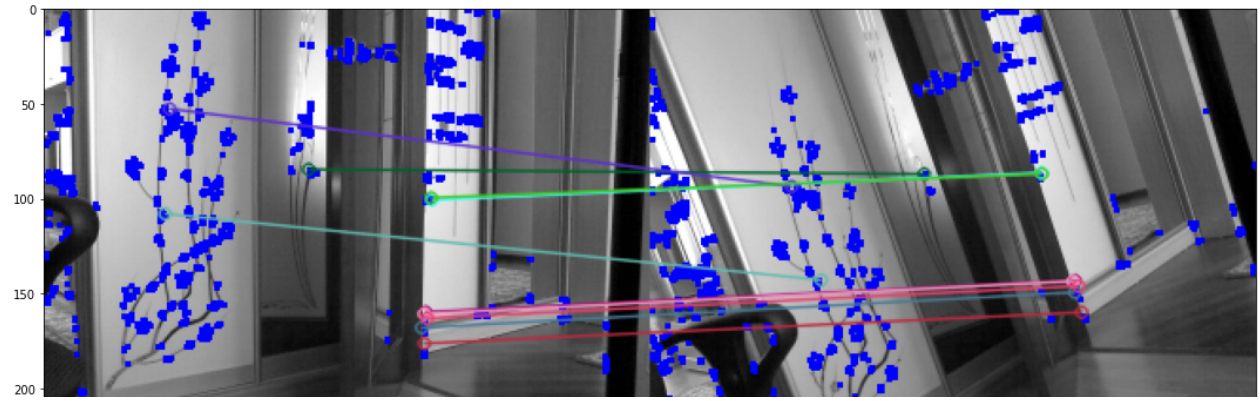



```
img2 = cv2.imread('img4_2.bmp')
gray = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)
dst2 = cv2.cornerHarris(gray,2,3,0.04)

# result is dilated for marking the corners, not important
dst2 = cv2.dilate(dst2,None)

# Threshold for an optimal value, it may vary depending on the image.
img2[dst2>0.01*dst2.max()]=[0,0,255]


plt.imshow(img2)
fig = plt.gcf()
fig.set_size_inches(10, 5)
```

```
plt.show()
```




```
# Initiate ORB detector
orb = cv2.ORB_create()

# find the keypoints and descriptors with ORB
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)

# create BFMatcher object
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

# Match descriptors.
matches = bf.match(des1,des2)

# Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)

# Draw first 10 matches.
img3 = cv2.drawMatches(img1,kp1,img2,kp2,matches[:10],None,flags=cv2.DrawMatchesFla
plt.imshow(img3)
fig = plt.gcf()
fig.set_size_inches(18.5, 10.5)
plt.show()
```




```
# Try to draw all matches.
img3 = cv2.drawMatches(img1,kp1,img2,kp2,matches,None,flags=cv2.DrawMatchesFlags_NO
plt.imshow(img3)
fig = plt.gcf()
fig.set_size_inches(18.5, 10.5)
plt.show()
```