|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

**Факультет «Информатика и системы управления»**

**Кафедра «Системы обработки информации и управления»**

Отчет по ДЗ

«*Синтез панорамного изображения путем склеивания*

*фрагментов из заданного набора*»

по дисциплине «Технология мультимедиа»

Выполнил:

студент группы РТ5-61Б Алиев Тимур

подпись, дата

Проверил:

к.т.н., доц., Г.И. Афанасьев

подпись, дата

2022 г.

# Оглавление

[Задание 3](#_Toc102491604)

[Описание алгоритма работы программы 3](#_Toc102491605)

[Текст программы 5](#_Toc102491606)

# Задание

Разработка программы для синтеза панорамного изображения путем склеивания фрагментов из заданного набора на языке Python с использованием библиотеки OpenCV.

# Описание алгоритма работы программы

Общий алгоритм выглядит следующим образом:

1. Найти похожие точки на двух рисунках и вычислить матрицу преобразования;

2. Преобразовать изображение и поместить его в подходящую позицию на другом изображении, и рассчитать новое значение пикселя перекрывающейся.

Теперь каждый пункт разберем подробно.

**1. Найти сходство**

SIFT – алгоритм для автоматического поиска сходств (соответствующие положения аналогичных точек). Библиотека OpenCV предоставляет интерфейс алгоритма SIFT.

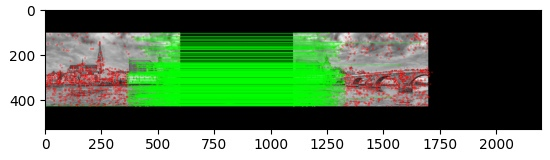
Ниже приведены исходные изображения двух тестовых изображений и изображений после нахождения сходства. Изображение выглядит как внешний, небо, старый

Автоматически созданное описание

Изображение 1. Левое.



Изображение 2. Правое.



Результат работы алгоритма SIFT.

Красные точки – это аналогичные точки (схожие), найденные алгоритмом SIFT, а зелёные линии представляют аналогичные точки с более высоким уровнем достоверности, выбранные среди всех найденных похожих точек. Потому что найденное алгоритмом сходство не обязательно на 100% правильно. Затем мы можем вычислить матрицу преобразований на основе выбранных сходств. OpenCV предоставляет нужный для этого интерфейс.

**2. Сшивание картинки**

Далее мы используем вычисленную матрицу преобразования для преобразования одного из изображений, затем наложим преобразованное изображение на другое изображение и пересчитаем новое значение пикселя перекрывающейся области. Для перекрывающейся области часть рядом с левой стороной позволяет левому изображению отображать больше, а часть рядом с правой стороной позволяет правому изображению отображать больше. Выражаясь формулой, предполагая, что альфа представляет расстояние от горизонтальной координаты точки пикселя до горизонтальной координаты левой и правой области перекрытия, новое значение пикселя равно newpixel = значение пикселя на левом изображении × (1-альфа) + значение пикселя на правом изображении × альфа. Таким образом, можно достичь простого эффекта слияния.

Изображение выглядит как текст, экран, снимок экрана, рама картины

Автоматически созданное описание

Результат работы программы.

# Текст программы

import numpy as np  
import cv2 as cv  
from matplotlib import pyplot as plt  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 top, bot, left, right = 100, 100, 0, 500  
 img1 = cv.imread('image/1.png')  
 img2 = cv.imread('image/2.png')  
 srcImg = cv.copyMakeBorder(img1, top, bot, left, right, cv.BORDER\_CONSTANT, value=(0, 0, 0))  
 testImg = cv.copyMakeBorder(img2, top, bot, left, right, cv.BORDER\_CONSTANT, value=(0, 0, 0))  
 img1gray = cv.cvtColor(srcImg, cv.COLOR\_BGR2GRAY)  
 img2gray = cv.cvtColor(testImg, cv.COLOR\_BGR2GRAY)  
 sift = cv.SIFT\_create()  
 # Поиск схожих точек с помощью SIFT  
 kp1, des1 = sift.detectAndCompute(img1gray, None)  
 kp2, des2 = sift.detectAndCompute(img2gray, None)  
 # FLANN параметры  
 FLANN\_INDEX\_KDTREE = 1  
 index\_params = dict(algorithm=FLANN\_INDEX\_KDTREE, trees=5)  
 search\_params = dict(checks=50)  
 flann = cv.FlannBasedMatcher(index\_params, search\_params)  
 matches = flann.knnMatch(des1, des2, k=2)  
 # Маска  
 matchesMask = [[0, 0] for i in range(len(matches))]  
  
 good = []  
 pts1 = []  
 pts2 = []  
  
 for i, (m, n) in enumerate(matches):  
 if m.distance < 0.7 \* n.distance:  
 good.append(m)  
 pts2.append(kp2[m.trainIdx].pt)  
 pts1.append(kp1[m.queryIdx].pt)  
 matchesMask[i] = [1, 0]  
  
 draw\_params = dict(matchColor=(0, 255, 0),  
 singlePointColor=(255, 0, 0),  
 matchesMask=matchesMask,  
 flags=0)  
 img3 = cv.drawMatchesKnn(img1gray, kp1, img2gray, kp2, matches, None, \*\*draw\_params)  
 plt.imshow(img3, ), plt.show()  
  
 rows, cols = srcImg.shape[:2]  
 MIN\_MATCH\_COUNT = 10  
 if len(good) > MIN\_MATCH\_COUNT:  
 src\_pts = np.float32([kp1[m.queryIdx].pt for m in good]).reshape(-1, 1, 2)  
 dst\_pts = np.float32([kp2[m.trainIdx].pt for m in good]).reshape(-1, 1, 2)  
 M, mask = cv.findHomography(src\_pts, dst\_pts, cv.RANSAC, 5.0)  
 warpImg = cv.warpPerspective(testImg, np.array(M), (testImg.shape[1], testImg.shape[0]),  
 flags=cv.WARP\_INVERSE\_MAP)  
  
 for col in range(0, cols):  
 if srcImg[:, col].any() and warpImg[:, col].any():  
 left = col  
 break  
 for col in range(cols - 1, 0, -1):  
 if srcImg[:, col].any() and warpImg[:, col].any():  
 right = col  
 break  
  
 res = np.zeros([rows, cols, 3], np.uint8)  
 for row in range(0, rows):  
 for col in range(0, cols):  
 if not srcImg[row, col].any():  
 res[row, col] = warpImg[row, col]  
 elif not warpImg[row, col].any():  
 res[row, col] = srcImg[row, col]  
 else:  
 srcImgLen = float(abs(col - left))  
 testImgLen = float(abs(col - right))  
 alpha = srcImgLen / (srcImgLen + testImgLen)  
 res[row, col] = np.clip(srcImg[row, col] \* (1 - alpha) + warpImg[row, col] \* alpha, 0, 255)  
  
 # opencv это bgr, matplotlib это rgb.  
 # преобразование из одного цветового пространства в другое  
 res = cv.cvtColor(res, cv.COLOR\_BGR2RGB)  
  
 # Вывод результата  
 plt.figure()  
 plt.imshow(res)  
 plt.show()  
 else:  
 print("Недостаточно совпадений - {}/{}".format(len(good), MIN\_MATCH\_COUNT))  
 matchesMask = None