Лабораторные работы по курсу

Моделирование ч.3 «Компьютерное зрение»

Лабораторная работа 3

Сшивка изображений

2021

Содержание

Введение

Целью лабораторной работы является освоение навыков программной реализации процедуры сшивки изображений с применением наиболее популярных алгоритмов в области компьютерного зрения, реализованных в библиотеке *OpenCV* [1].

Лабораторная работа содержит два задания для самостоятельного выполнения. Первое задание заключается в реализации процедуры сшивки двух изображений. Второе задание заключается в исследовании быстродействия процедуры сшивки изображений в зависимости от выбранного подхода в решении задачи поиска попарного соответствия особых точек.

Лабораторная работа рассчитана на 4 академических часа.

1. Краткая теоретическая справка

1.1. Общая последовательность действий для сшивки изображений

Сшивка изображений представляет собой процесс объединения нескольких снимков, содержащих одинаковые объекты, в единое изображение большего размера. Сшивка изображений является основой для создания панорам и широко используется в картографии, астрономии, медицине, производстве, беспилотном транспорте и т. д.

Общая последовательность действий, необходимых для сшивки изображений, представлена на Рисунке Рисунок 1.

Рисунок 1. Процедура сшивки изображений

Первым этапом сшивки изображений является поиск характерных признаков на каждом изображении. Характерные признаки разделяются на контуры объектов (*lines, edges*), области (*blobs*) и ключевые точки (*keypoints*). Ввиду наибольшей универсальности, современные алгоритмы поиска характерных особенностей, как правило, оперируют ключевыми точками. Для каждой ключевой точки совместно с ее локализацией на изображении должно быть сформировано описание (дескриптор), по которому данную ключевую точку можно искать на других изображениях. В общем случае, в качестве описания ключевой точки может выступать ее локальная окрестность небольшого размера (8×8…16×16 пикс.).

Вторым этапом является поиск взаимно-однозначного соответствия между одинаковыми ключевыми точками на разных изображениях. Степень схожести ключевых точек определяется на основе сравнения их дескрипторов с помощью одной из метрик, например, кросс-корреляции, свертки, суммы разницы квадратов локальных окрестностей и т. д.

Третьим этапом является поиск матрицы гомографии между изображениями. Объекты на каждом снимке подвергаются перспективным искажениям, которые зависят от положения и ориентации камеры в пространстве. Для совмещения всех изображений в одной плоскости необходимо предварительное устранение перспективных искажений.

Заключительным этапом является применение матриц гомографии к изображениям и формирование единого двумерного массива, состоящего из преобразованных входных изображений.

1.2. Алгоритмы поиска ключевых точек

На сегодняшний день существует достаточно большое количество различных алгоритмов поиска ключевых точек на изображениях (*keypoint detection*). Одним из наиболее известных и простых является детектор Харриса [2], опубликованный в 1988 году. Рассмотрим современные алгоритмы.

1.2.1. Алгоритм *FAST*

Алгоритм *FAST* (*Features from Accelerated Segment Test* – особенности ускоренных испытаний сегмента) был предложен в 2005 г. Эдвардом Ростеном и Томом Драммондом [3].

Для поиска особых точек вокруг каждого пикселя изображения рассматривается окружность из 16 пикселов (Рисунок Рисунок 2). Если на окружности встречается N подряд идущих пикселей с яркостью $cp_i > p + t$ или $cp_i < p + t$, где cp_i – яркость i-ого пиксела на окружности, p – яркость центрального пиксела, t – пороговая величина, то центральный пиксел помечается как особая точка. Исследования, проведенные авторами, показали, что для стабильного обнаружения особых точек, N должно быть равно 9. Для повышения быстродействия алгоритма первично проверяются только уровни яркости в пикселах с номерами 1, 5, 9, 13, а только затем рассматриваются остальные пикселы на окружности.

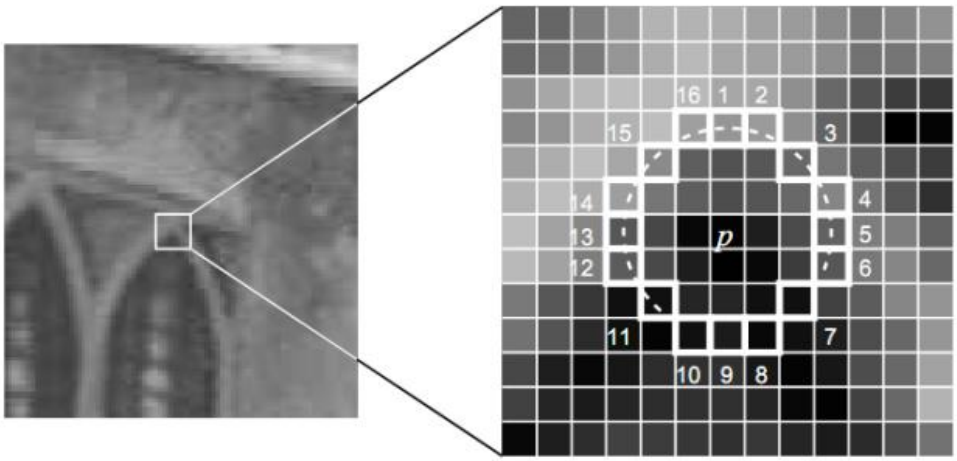

Рисунок 2. Иллюстрация к алгоритму FAST

Достоинством алгоритма *FAST* является высокая скорость работы за счет малого количества операцией. Недостатком является обнаружение нескольких особых точек внутри одной локальной окрестности, вследствие чего эффективность работы алгоритма зависит от порядка обработки изображения.

Алгоритм *FAST* реализован в библиотеке компьютерного зрения *OpenCV* [4] и может быть использован как

```
fast = cv2.FastFeatureDetector_create()
keypoints = fast.detect(img, None)
```

где *img* – входное изображение.

Однако в алгоритме *FAST* реализован только поиск ключевых точек, но не реализовано формирование их описания. Для получения описания можно воспользоваться функцией другого детектора. Например, *SIFT*:

```
fast = cv2.FastFeatureDetector_create()
sift = cv2.SIFT_create()
keypoints = fast.detect(img, None)
descriptors = sift.compute(img, keypoints)[1]
```

1.2.2. Алгоритм *SIFT*

Алгоритм *SIFT* (*scale-invariant feature transform* – масштабно-инвариантная трансформация признаков) является запатентованным алгоритмом, опубликованным Девидом Лоу в 1999 году [5].

Основа поиска особых точек в алгоритме *SIFT* базируется на построении пирамиды разномасштабных изображений, где в основании пирамиды располагается изображение исходного масштаба, а каждое следующее изображение уменьшается в 2 раза (Рисунок Рисунок 3). Изображения промежуточного масштаба формируются с помощью интерполяции из двух изображений соседнего масштаба. Изображения, масштаб которых кратен двум, называются октавами.

После процедуры построения пирамиды между соседними изображениями считается разница, тем самым формируя новую пирамиду приращений яркости от масштаба к масштабу.

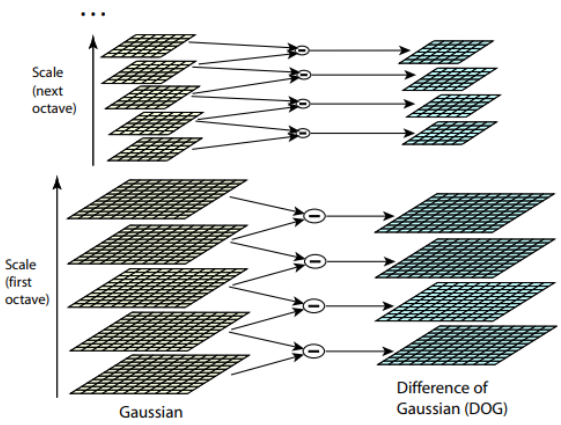

Рисунок 3. Пирамида изображений в алгоритме SIFT

За особую точку в пирамиде приращений изображений принимается такая точка, которая является локальным минимум или максимумом в кубе размером 3×3 (Рисунок Рисунок 4).

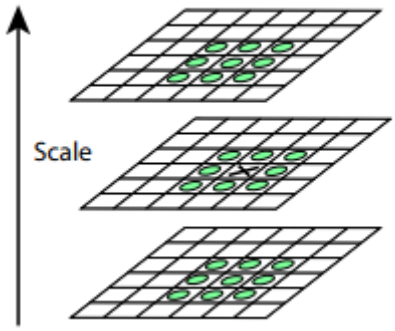

Рисунок 4. Локальная окрестность особой точки в алгоритме SIFT.

Достоинством алгоритма *SIFT* является его устойчивость к изменению масштаба особых точек на изображении. К недостаткам можно отнести наличие патента, который требует получение лицензии для применения данного алгоритма в коммерческом использовании.

Алгоритм *SIFT* реализован в библиотеке компьютерного зрения *OpenCV* [6] и может быть использован как

```
sift = cv2.SIFT_create()
keypoints, descriptors = sift.detectAndCompute(img, None)
```

где *img* – входное изображение, *keypoints* – набор ключевых точек, *descriptors* – описания ключевых точек.

1.2.3. Алгоритм *ORB*

Алгоритм *ORB* (*Oriented FAST and Rotated BRIEF*) был предложен в 2011 году [7] как открытая альтернатива запатентованным алгоритмам поиска особых точек. Общая идея алгоритма заключается в вычислении ориентации особой точки и проведения сравнений на ее основе.

Всего можно выделить три шага в *ORB*. На первом шаге особые точки обнаруживаются с помощью улучшенного древовидного алгоритма FAST [8] на исходном изображении и на нескольких уменьшенных копиях изображения. Затем для подтверждения особых точек вычисляется мера Харриса [2] и точки с малым значением отбрасываются путем порогового сравнения. Затем для каждой найденной особой точки вычисляется ее угол ориентации, который ложится в основу дескриптора.

Алгоритм *ORB* реализован в библиотеке компьютерного зрения *OpenCV* [9] и может быть использован как

```
orb = cv2.ORB_create()
keypoints, descriptors = orb.detectAndCompute(img, None)
```

где *img* – входное изображение, *keypoints* – набор ключевых точек, *descriptors* – описания ключевых точек.

1.3. Алгоритмы сопоставления ключевых точек

Для сопоставления двух изображений необходимо найти взаимно-однозначное соответствие их ключевых точек (*matching*). Рассмотрим два подхода к решению данной задачи, реализованных в библиотеке *OpenCV*.

1.3.1. Полный попарный перебор

Полный попарный перебор всех возможных комбинаций ключевых точек является гарантированным методом определения наиболее вероятного взаимно-однозначного соответствия (Рисунок Рисунок 5).

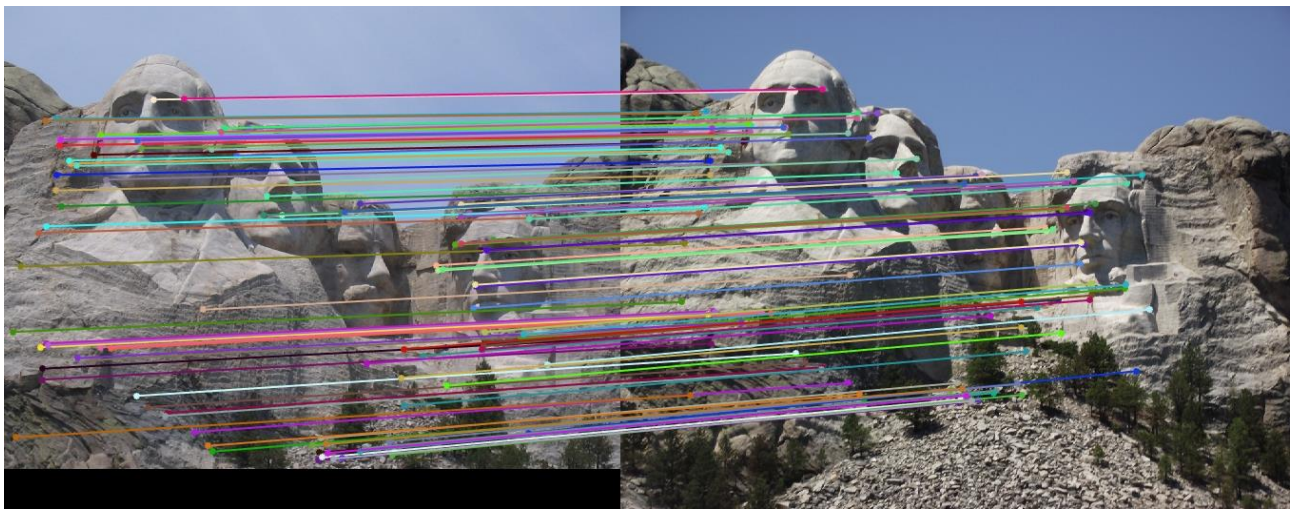

Рисунок 5. Визуализация взаимно-однозначного соответствия особых точек

В библиотеке *OpenCV* данный подход реализован в классе *BFMatcher* [10] и может быть использован как

```
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1, des2, k=2)
```

где *des1* и *des2* – дескрипторы особых точек на двух изображениях. Параметр *k* определяет количество выдаваемых наиболее близких особых точек из *des2* для каждой

особой точки из *des1*. Например, если *k=2*, то в выходном списке соответствий для каждой особой точки из *des1* будет представлено две точки из *des2* с наиболее близким дескриптором. Данную информацию можно учитывать путем сравнения двух расстояний и аннулирования соответствия, если удаленности дескриптора особой точки схожи между собой по величине, тем самым отбрасывая ложные совпадения.

Недостатком полного перебора попарного соответствия особых точек является большая вычислительная сложность, которая пропорциональна квадрату их количества.

1.3.2. Оптимизация перебора по скорости

В библиотеке *OpenCV* реализован класс *FlannBasedMatcher* [11], с помощью которого можно повысить скорость решения задачи поиска попарного соответствия. Основой данного подхода является рандомизация выбора пары особых точек для вычисления расстояния между дескрипторами, которая статистически будет сходиться к оптимальной. С помощью данного метода можно достаточно быстро найти попарное соответствие, однако поскольку проверяются не все возможные варианты, существует вероятность, что выданный результат в конкретном случае не будет наилучшим из возможных.

Данный подход может быть использован как

```
flann = cv2.FlannBasedMatcher_create()
matches = flann.knnMatch(des_r, des_l, k=2)
```

где *des1* и *des2* – дескрипторы особых точек на двух изображениях

1.4. Наложение изображений

1.4.1. Матрица гомографии

Гомография представляет собой частный случай модели фундаментального проективного преобразования, когда все объекты сцены находятся в одной плоскости (Рисунок Рисунок 6), а матрица гомографии $\mathbf{H}$ описывает переход от одной плоскости к другой.

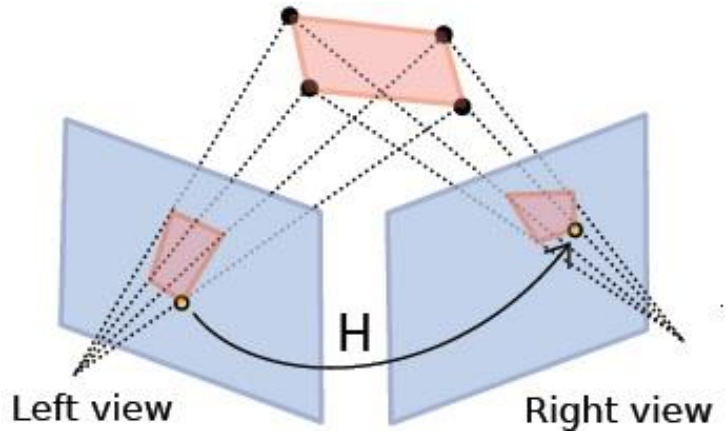

Рисунок 6. Гомография

Общее проективное преобразование описывается матричным уравнением

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K} \cdot \begin{bmatrix} \mathbf{R} & \mathbf{T} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_{03} \\ r_{10} & r_{11} & r_{12} & t_{13} \\ r_{20} & r_{21} & r_{22} & t_{23} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix},$$

где

- $(u, v, 1)$ – положение точки в координатах изображения (пикселы), записанное в однородной форме,
- (x, y, z) – положение точки в пространстве в мировой системе координат (X, Y, Z),
- $\mathbf{K}$ – матрица внутренних параметров камеры размером 3×3,
- $\begin{bmatrix} \mathbf{R} & \mathbf{T} \end{bmatrix}$ – матрица поворота и переноса мировой системы координат (X, Y, Z) в систему координат камеры,
- f_x, f_y – фокусное расстояние камеры,
- c_x, c_y – координаты центра оптической оси камеры,
- r_{ij} и t_{i3} – коэффициенты матриц $\mathbf{R}$ и $\mathbf{T}$ соответственно.

За счет того, что объекты лежат в одной плоскости, координата *z* равна нулю и общее проективное преобразование записывается как

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{00} & r_{01} & t_{03} \\ r_{10} & r_{11} & t_{13} \\ r_{20} & r_{21} & t_{23} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix},$$

где h_{ij} – коэффициенты матрицы гомографии $\mathbf{H}$.

Если все коэффициенты матрицы $\mathbf{H}$ разделить на h_{22}, то итоговые координаты могут быть вычислены как

$$\begin{aligned} u &= \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + 1} \\ v &= \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + 1} \end{aligned},$$

а матрица гомографии $\mathbf{H}$ будет состоять из 8 независимых коэффициентов:

$$\mathbf{H} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix}.$$

Следовательно, для однозначного вычисления всех коэффициентов матрицы $\mathbf{H}$ необходимо иметь 4 попарных соответствия между особыми точками двух изображений.

1.4.2. Вычисление и применение матрицы гомографии

Ввиду того, что количество находимых попарно соответствующих друг другу ключевых точек при сшивке изображений, как правило, превышает 4, матрица гомографии может быть рассчитана исходя из переопределенной системы линейных уравнений с помощью различных методов аппроксимации.

В библиотеке *OpenCV* для поиска матрицы гомографии между двумя изображениями реализована функция *findHomography* [12], которая может быть использована как

```
H = cv2.findHomography(srcPoints, dstPoints, method, ransacReprojThreshold)
```

где *srcPoints* – массив координат ключевых точек на одном изображении, *dstPoints* – массив координат ключевых точек на втором изображении, *method* – метода решения переопределенной системы линейных уравнений, *ransacReprojThreshold* – порог максимально допустимой ошибки в отклонении проективного преобразования между парой точек.

В качестве метода аппроксимации может быть выбран *RANSAC* [13], *LMEDS* [14] или *RHO*.

Применение матрицы гомографии $\mathbf{H}$ к изображению может быть выполнено с помощью функции *warpPerspective* [15] библиотеки *OpenCV* как:

```
Img_out = cv2.warpPerspective(img_in, H, dsize)
```

где *img_in* – исходное изображение, *H* – матрица гомографии, *dsize* – размеры результирующего изображения.

2. Упражнения

2.1. Подготовка

2.1.1. Сделайте две произвольные фотографии с перекрывающимися областями по аналогии с Рисунком Рисунок 7.

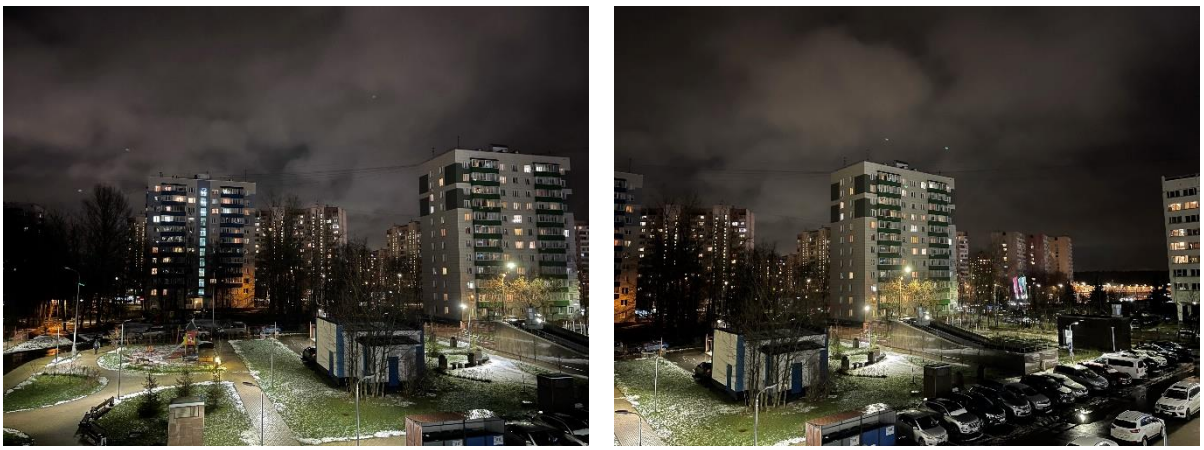

Рисунок 7. Входные изображения (слева left.jpg, права right.jpg)

2.1.2. Создайте директорию «*lab3*» и создайте в ней поддиректории:

- *lab3/data*
- *lab3/panorama*

2.1.3. Поместите сделанные изображения в директорию «*lab3/data*». Присвойте снимкам наименования «*left.jpg*» и «*right.jpg*» соответственно.

2.1.4. Откройте *PyCharm IDE* и создайте новый проект с названием «*panorama*» в директории «*lab3/panorama*».

2.1.5. Установите в виртуальном окружении проекта следующие библиотеки:

- *opencv-python;*
- *numpy;*
- *matplotlib.*

2.2. Разработка программы

2.2.1. В файле «*main.py*» подключите библиотеки к проекту:

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

2.2.2. Прочитайте оба изображения и преобразуйте их в полутоновые изображения:

```
img_right_rgb = cv2.imread("../data/right.jpg")
img_right_gray = cv2.cvtColor(img_right_rgb, cv2.COLOR_BGR2GRAY)
img_left_rgb = cv2.imread("../data/left.jpg")
img_left_gray = cv2.cvtColor(img_left_rgb, cv2.COLOR_BGR2GRAY)
```

2.2.3. Найдите ключевые точки и их дескрипторы для каждого изображения с помощью алгоритма *FAST* их описания с помощью алгоритма *SIFT*:

```
fast = cv2.FastFeatureDetector_create()
sift = cv2.SIFT_create()
kp_r = fast.detect(img_right_gray, None)
kp_l = fast.detect(img_left_gray, None)
des_r = sift.compute(img_right_gray, kp_r)[1]
des_l = sift.compute(img_left_gray, kp_l)[1]
```

2.2.4. Найдите для каждой ключевой точки из одного изображения две наиболее подходящие ключевые точки другого изображения. Выберите для каждой ключевой точки одного изображения только одно совпадение с помощью фильтрации (переменная «*dist_ratio*» задает минимально допустимое отношение расстоянии между дескрипторами для того, чтобы взаимно-однозначное соответствие между точками считалось хорошим):

```
bf = cv2.BFMatcher()
matches = bf.knnMatch(des_r, des_l, k=2)

good = []
dist_ratio = 0.9
for m in matches:
    if (m[0].distance/m[1].distance) < dist_ratio:
        good.append(m)
matches = np.asarray(good)
```

2.2.5. Найдите матрицу гомографии между ключевыми точками. Обратите внимание, что количество взаимно-однозначных совпадений должно быть не менее четырех.

```
if len(matches[:,0]) < 4:
    raise AssertionError("Homography is not available. Number of keypoints less
than 4")

left = np.float32([kp_r[m.queryIdx].pt for m in matches[:, 0]]).reshape(-1, 1,
2)
right = np.float32([kp_l[m.trainIdx].pt for m in matches[:, 0]]).reshape(-1, 1,
2)
H, mask = cv2.findHomography(left, right, cv2.RANSAC, 5.0)
```

2.2.6. Преобразуйте правое изображение с помощью матрицы гомографии «*H*» и сшейте его с левым изображением:

```
img_total = cv2.warpPerspective(img_right_rgb, H, (img_right_rgb.shape[1] +
img_left_rgb.shape[1], img_left_rgb.shape[0]))
img_total[0:img_left_rgb.shape[0], 0:img_left_rgb.shape[1]] = img_left_rgb
```

2.2.7. Отобразите результирующее изображение (Рисунок Рисунок 8):

```
plt.figure()
plt.title('total image')
plt.imshow(img_total)
plt.show()
```

Рисунок 8. Результирующее изображение

3. Подготовка к выполнению лабораторной работы

3.1. Изучить теоретический материал, представленный в лабораторном практикуме.
3.2. Изучить дополнительную литературу, представленную в разделе п. 5 «Список литературы» лабораторного практикума.
3.3. Выполнить п. 2 «Упражнения» лабораторного практикума.

4. Индивидуальные задания

4.1. Задание №1

4.1.1. Реализовать процедуру сшивки двух изображений с помощью функций, доступных в библиотеке OpenCV. Алгоритм поиска особых точек выбирается в зависимости от

$$i = (N \,\%\, 2),$$

где N – ваш номер в общем списке группы:

- $i = 1$: алгоритм *SIFT*
- $i = 2$: алгоритм *ORB*

4.1.2. Процедура сшивки должна завершаться выводом единого изображения с помощью библиотеки *matplotlib*.
4.1.3. Предусмотреть промежуточный вывод двух изображений с отображением попарного совпадение ключевых точек.

4.2. Задание №2

4.2.1. Провести измерение и сравнение времени работы процедуры сшивки изображений, разработанной в Задании №1, при использовании класса *BFMatcher* и класса *FlannBasedMatcher*.

5. Список литературы

1. OpenCV [Электронный ресурс] // OpenCV: [сайт]. URL: https://opencv.org/
2. Chris Harris M.S. A combined corner and edge detector, 1988.
3. Drummond E.R.A.T. Fusing Points and Lines for High Performance Tracking, 2005.
4. FAST Algorithm for Corner Detection [Электронный ресурс] // OpenCV-Python-Tutorials: [сайт]. URL: https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_fast/py_fast.html
5. Lowe D.G. Object recognition from local scale-invariant features // Proceedings of the International Conference on Computer Vision, 1999. P. 1150—1157.
6. Introduction to SIFT (Scale-Invariant Feature Transform) [Электронный ресурс] // OpenCV: [сайт]. URL: https://docs.opencv.org/3.4/da/df5/tutorial_py_sift_intro.html
7. Ethan Rublee V.R.K.K.G.B. ORB: an efficient alternative to SIFT or SURF // Computer Vision (ICCV), IEEE International, 2011. pp. 2564 – 2571.
8. Edward Rosten T.D. Machine Learning for High-speed Corner Detection // Computer Vision – ECCV 2006, 2006. pp. 430-443.
9. ORB (Oriented FAST and Rotated BRIEF) [Электронный ресурс] // OpenCV-python Tutorials: [сайт]. URL: https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_orb/py_orb.html
10. cv:BFMatcher Class Reference [Электронный ресурс] // OpenCV documentation: [сайт]. URL: https://docs.opencv.org/3.4/d3/da1/classcv_1_1BFMatcher.html
11. Feature Matching with FLANN [Электронный ресурс] // OpenCV documentation: [сайт]. URL: https://docs.opencv.org/3.4/d5/d6f/tutorial_feature_flann_matcher.html
12. Camera Calibration and 3D Reconstruction | findHomography() [Электронный ресурс] // OpenCV documentation: [сайт]. URL: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga4abc2ece9fab9398f2e560d53c8c9780
13. Martin A. Fischler R.C.B. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography // Comm. Of the ACM : journal, 1981. pp. 381-395.
14. LMEDS: Least Median of Squares [Электронный ресурс] URL: http://www-sop.inria.fr/odyssee/software/old_robotvis/Tutorial-Estim/node25.html
15. Geometric Image Transformations | warpPerspective [Электронный ресурс] // OpenCV documentation: [сайт]. URL: https://docs.opencv.org/4.x/da/d54/group__imgproc__transform.html#gaf73673a7e8e18ec6963e3774e6a94b87