

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Отчет по лабораторной работе №3
«Детекторы характеристических точек»
по дисциплине «Техническое зрение»

Выполнил: студент гр. Р3338,

Кирбаба Д.Д.

Преподаватель: Шаветов С.В.,

канд. техн. наук, доцент ФСУ и Р

Санкт-Петербург, 2022

Цель работы

Исследование детекторов и дескрипторов характеристических точек.

Теоретическое обоснование применяемых методов

В компьютерном зрении характеристическая точка есть часть информации о содержании изображения; обычно о том, обладает ли определенная область изображения определенными свойствами.

Свойства характеристических точек:

- В окрестности характеристической точки должен содержаться *rich image content* (вариации яркости, цвета...), который предписывает некоторое количество уникальности данной области. Впоследствии такая информация может быть использована для задач сопоставления характеристических точек.
- Должна иметь строго-определенное описание (сигнатуру) для задач сопоставления/сравнения с другими характеристическими точками.
- Должна иметь определенную позицию на изображении.
- Должна быть инвариантна к повороту и масштабированию.
- Должна быть нечувствительна к изменениям освещения.

Типы характеристических точек:

- Edges
- Corners
- Blobs
- Ridges

В некоторых случаях недостаточно выделить только один тип характеристических точек, чтобы получить соответствующую информацию из данных изображения. Вместо этого извлекаются два или более различных признака, в результате чего в каждой точке изображения появляется два или более дескриптора признаков. Общепринятой практикой является организация информации, предоставляемой всеми этими дескрипторами, в виде элементов одного единственного вектора, обычно называемого вектором признаков.

Для того, чтобы извлечь из изображения признаки, используются детекторы характеристических точек, они содержат методы для вычисления абстракций информации об изображении и принятия локальных решений в каждой точке изображения, есть ли в этой точке признак данного типа или нет.

При работе с характеристическими точками обычно используется следующая последовательность действий:

1. С помощью детектора находятся характеристические точки изображения.
2. С помощью дескриптора каждой характеристической точкедается сигнтура.
3. Проводятся необходимые задачи по сопоставлению, сравнению, поиску и т. д.

Извлечение признаков иногда производится по нескольким масштабам. Одним из таких методов является масштабно-инвариантное преобразование признаков (SIFT).

SIFT

SIFT успешно используется для 2D распознавания объекта, выравнивания изображений а также для склеивания изображений.

Для решения проблемы различных масштабов характеристических точек используется представление изображения в *scale – space* – это однопараметрическое семейство сглаженных фильтром Гаусса изображений (в качестве параметра выступает дисперсия σ).

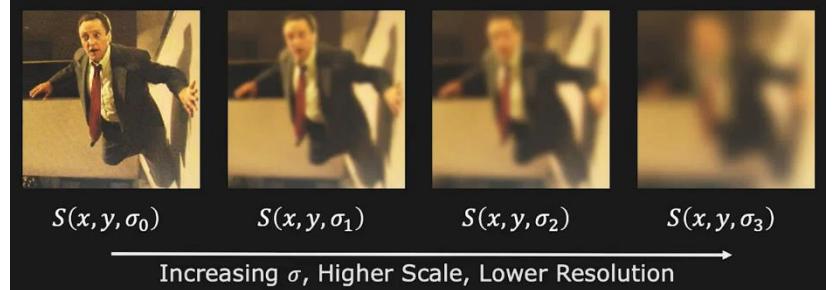


Figure 1: Scale – space example.

Для расчета масштаба характеристических точек используется метод *LoG* (*Laplassian of Gaussian*). Его можно рассчитать как максимальный отклик *LoG* изображения в пространстве *scale – space* при изменении значения σ .

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Для эффективного определения максимального отклика используется *DoG* (*Difference of Gaussians*):

$$D(x, y, \sigma) = (G(x, y, \sigma) - G(x, y, k\sigma)) * I(x, y)$$

$$= L(x, y, \sigma) - L(x, y, k\sigma)$$

Так как $DoG = L(x, y, \sigma) - L(x, y, k\sigma) \approx (k - 1)L(x, y, \sigma)$, то мы можем не вычислять лапласианы, а получить серию изображений с пиками из разности ближайших изображений в *scale – space*.

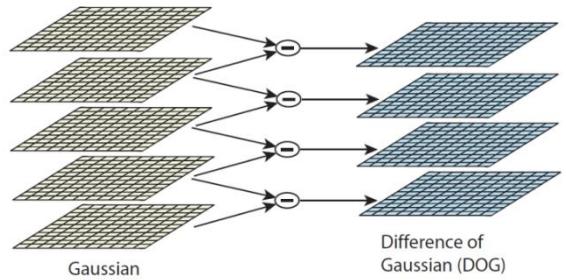


Figure 2: вычисление DoG.

Максимумы свертки *DoG* для пикселя могут быть рассчитаны сравнением пикселя с его 26 соседями в текущем и соседних масштабах (в кубе $3x3x3$).

В итоге мы получим серию изображений с вероятными характеристическими точками различных масштабов. Для удаления слабых экстремумов необходимо применить пороговую фильтрацию.

Опишем задачу поиска фрагмента изображения на другом изображении.

Найдем по вышенаписанному алгоритму характеристические точки на обоих изображениях вместе с их масштабом.

Тогда для приведения масштаба одного изображения к другому, необходимо выбрать характеристическую точку, присутствующую на обоих изображениях.

Если масштаб изображений различный, значения $\sigma_1 \neq \sigma_2$, для приведения к одному масштабу необходимо увеличить в $k: k\sigma_1 = \sigma_2$ раз изображение.

Также на двух изображениях может не совпадать ориентация. Для приведения ориентации в соответствие необходимо вычислить гистограмму направлений градиентов в областях с характеристическими точками.

Формула направлений градиентов:

$$\theta = \tan^{-1}(\frac{\delta I}{\delta y} / \frac{\delta I}{\delta x})$$

Так как мы используем только направления градиента, а модуль градиента не учитываем, то наш алгоритм становится инвариантен к изменениям освещения (так как величина модуля градиента как раз соответствует изменению освещения изображения).

Итак, теперь построим гистограмму по направлениям, присущим в вычисляемой области характеристической точки.

И найдя максимальное значение гистограммы, соответствующее направление будет основным для данной характеристической точки.

Всё что остается сделать для приведения изображений к ориентации – повернуть одно из них к соответствующему основному направлению.

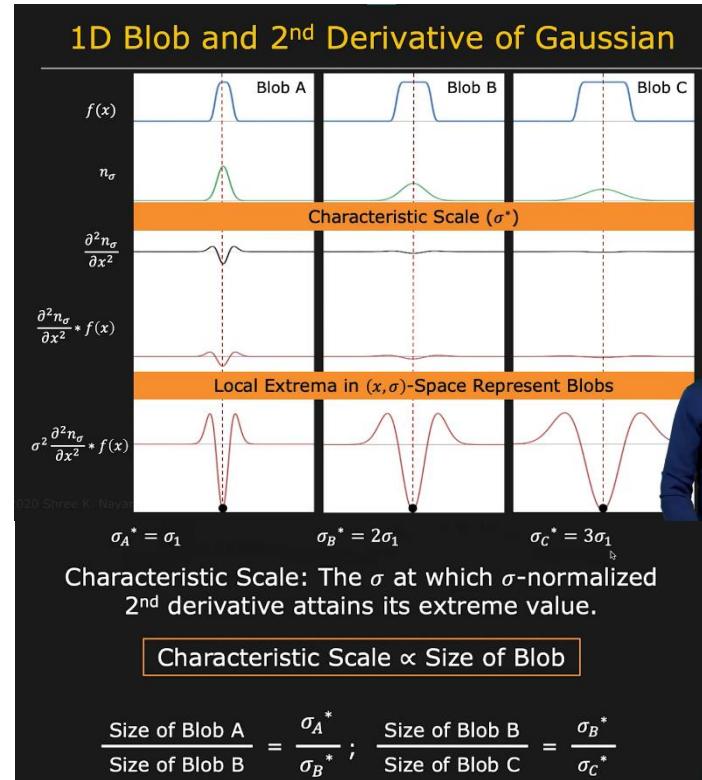


Figure 3: пример поиска масштаба blobs.

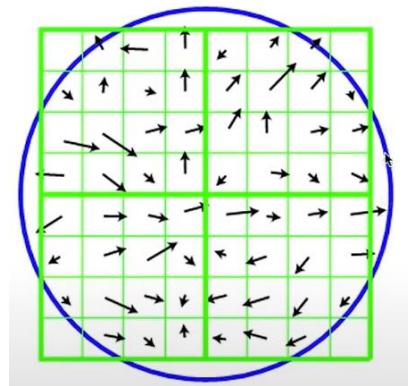


Figure 4: область с направлениями градиентов.

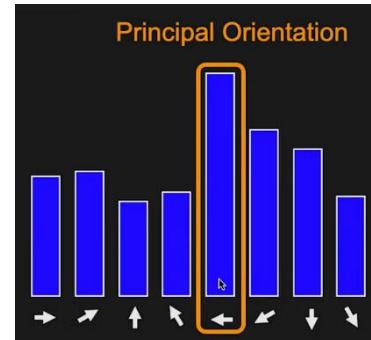


Figure 5: гистограмма направлений градиента.

одной

Теперь вернемся к вопросу сопоставления характеристических точек. Для поиска соответствующих точек на изображениях необходимо каким-то образом описать точку (дать ей сигнатуру). Для этого используем SIFT дескриптор.

Для описания характеристической точки вначале необходимо вычислить её масштаб и основное направление, затем повернуть область точки, чтобы основное направление смотрело вверх. Затем для каждого квадранта из области направления градиентов характеристической точки построим гистограмму направлений.

В итоге, построенная гистограмма инвариантна к масштабу повороту и освещению изображения.

Поэтому её можно успешно использовать в качестве сигнатуры характеристической точки.

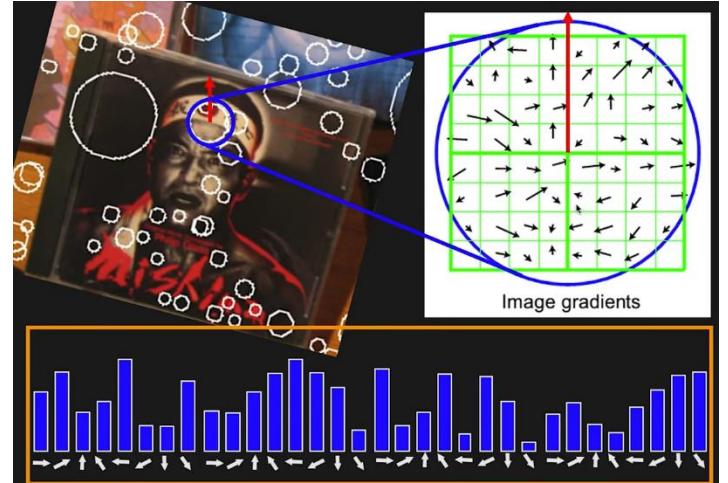


Figure 6: дескриптор SIFT.

Сравнивать сигнатуры характеристических точек можно двумя способами:

- Используя L_2 норму: $d(H_1, H_2) = \sqrt{\sum_k (H_1(k) - H_2(k))^2}$
- Используя нормализованную корреляцию: $d(H_1, H_2) = \frac{\sum_k [(H_1(k) - \bar{H}_1)(H_2(k) - \bar{H}_2)]}{\sqrt{\sum_k (H_1(k) - \bar{H}_1)^2} \sqrt{\sum_k (H_2(k) - \bar{H}_2)^2}}$
- *Intersection metric*: $d(H_1, H_2) = \sum_k \min(H_1(k), H_2(k))$

Стоит также отметить, что SIFT детектор можно применять только при малых изменениях точки наблюдателя (*viewpoint*).

ORB

Детектор ORB представляет собой слияние детектора характеристических точек FAST и BRIEF дескриптор с множеством модификаций для улучшения работы детектора.

Сначала он использует FAST-детектор для поиска характеристических точек, затем применяет угловую меру Харриса, чтобы найти среди них самые «важные» N -точки.

Поскольку FAST детектор не инвариантен к вращению, следующий метод используется для расчета вращения точки:

- Рассчитывается интенсивность взвешенного центроида области характеристической точки с расположенным углом в центре.
- Затем направление вектора от этой угловой точки к центроиду считается за ориентацию характеристической точки.

Чтобы улучшить инвариантность к вращению, моменты вычисляются с помощью осей x и y , которые должны находиться в круговой области радиуса r , где r — размер области характеристической точки.

Характеристические точки описываются BRIEF дескриптором. BRIEF дескриптор – это описание битовой строкой области характеристической точки, построенный на множестве бинарных тестов интенсивностей:

$$\tau(p, x, y) = \begin{cases} 1, & p(x) < p(y), \\ 0, & p(x) \geq p(y). \end{cases}$$

Расчет бинарной строки производится следующим образом:

$$f_n(p) = \sum_{i=1}^n 2^{i-1} \tau(p, x_i, y_i).$$

Чтобы повысить производительность дескриптора BRIEF для повернутых точек, дескриптор поворачивается в соответствии с ориентацией таких точек.

Для любого набора функций бинарных тестов n на месте (x_i, y_i) , определяется матрица $S_{2 \times n}$, содержащая координаты этих пикселей.

$$S = \begin{pmatrix} x_1 & \dots & x_n \\ y_1 & \dots & y_n \end{pmatrix}$$

Затем, используя ориентацию θ области, его матрица вращения R_θ рассчитывается и используется для поворота матрицы S , чтобы получить повернутую версию S_θ .

$$S_\theta = R_\theta S$$

ORB квантует угол с шагом $\frac{2\pi}{30} = 12$ градусов, поэтому LUT – таблица с помощью предварительно вычисленных шаблонов BRIEF может быть рассчитана для каждого возможного угла. Пока ориентация характеристической точки θ постоянна между точками обзора правильный набор точек S_θ будет использоваться для вычисления его дескриптора.

$$g_n(p, \theta) = f_n(p) | (x_i, y_i) \in S_\theta$$

Для вычисления расстояния между двумя дескрипторами ORB, можно использовать расстояние Хэмминга.

А для сопоставления сигнатур характеристических точек используется *multi – probe Locality – sensitive hashing (LSH)*.

Сопоставление сигнатур характеристических точек

Простейший способ сопоставить сигнатуры точек — это метод перебора.

В этом случае для каждой точки первого набора выбирается элемент второго набора, имеющий наименьшее расстояние. Для дескриптора SIFT используется евклидово расстояние L_2 . А для

дескриптора ORB, поскольку он является бинарной маской, используют расстояние Хэмминга между дескрипторами характеристических точек.

Очевидно, что простой перебор работает медленно. Для его ускорения поверх набора дескрипторов должна быть построена некоторая ускоряющая структура. Которая при сопоставлении дескриптора будет сравнивать его не со всем множеством других дескрипторов, а с некоторым подмножеством (кластером).

Простейший пример такой ускоряющей структуры - *KD-деревья* (*k – dimensional trees*), которая строится на тренировочном множестве дескрипторов. Если дескриптор представлен в виде бинарной строки, то лучше использовать метод Locality – Sensitive Hashing (LSH).

Использование первого наилучшего совпадения может привести к множеству совпадений дескрипторов, однако многие из них являются ложными совпадениями из-за того, что некоторые характеристические точки взяты из повторяющегося узора (окна, вода, облака и т. д.).

Есть два возможных решения для фильтрации некоторых слабых совпадений.

Первое решение — использовать перекрестную проверку, которая требует, чтобы дескриптор сопоставлялся в двух направлениях: при сопоставлении двух изображений он должен быть наилучшим совпадением как в прямом, так и в обратном направлениях.

Второе решение — использовать метод *k – nearest* соответствия. В этом случае для каждой точки находится несколько лучших совпадений, отсортированных по расстоянию, а совпадение считается «хорошим», если оно существенно отличается от значения следующего ближайшего совпадения, поэтому расстояние между первым ближайшим совпадением значительно ниже по сравнению с расстоянием с вторым ближайшим совпадением:

$$D_1 < r \cdot D_2$$

, где D_1 — расстояние до первого ближайшего совпадения, D_2 — расстояние до второго ближайшего совпадения, а r — коэффициент различия, который, по рекомендациям авторов метода SIFT должен быть 0.75.

Следует помнить, что метод сопоставления *k – nearest* несовместим с перекрестной проверкой, поскольку перекрестная проверка не позволяет найти более одного совпадения дескрипторов.

Используя ускоряющие структуры и *k – nearest* фильтрацию, мы смогли получить набор сильных совпадений между изображениями. Поскольку при сопоставлении дескрипторов мы не учитывали положение характеристических точек, то следующим шагом будет вычисление геометрического преобразования между изображениями, принимая во внимание, что все еще может быть много выбросов или ложных совпадений.

Наиболее используемое решение для этого метод *RANSAC* – Random Sequence Consensus.

Общая идея метода заключается в том, чтобы оценить не все данные, а только небольшую выборку, затем построить гипотезу на основе этой выборки и проверить, насколько эта гипотеза верна. После проверки нескольких таких гипотез мы выбираем одну, которая наилучшим образом согласуется с основными данными.

Распишем шаги данного метода:

1. На входе у нас есть множество пар сопоставляемых координат характеристических точек двух изображений: $S = \{(x, y) | x \in X, y \in Y\}$, где X – первое изображение, Y – второе.
2. Для каждого i от 1 до N строим гипотезы и проверяем их:
 - a. Строим гипотезу θ_i выбирая случайно пары $S_i = \{(x_i, y_i) | x_i \in X, y_i \in Y\} \in S$. В нашем случае достаточно выбрать 4 точки из каждого изображения для построения матрицы M для оценки нашей гипотезы.
 - b. Оценим гипотезу θ_i применив матрицу M ко всем точкам из множества характеристических точек первого изображения X и проверим их соответствия с точками из второго множества Y с применением некоторого порогового значения. Число соответствий — это оценка нашей гипотезы $R(\theta_i)$:

$$R(\theta) = \sum_{x \in X} p(\theta, x, Y)$$

$$p(\theta, x, Y) = \begin{cases} 1, & |\varepsilon(\theta, x, Y)| \leq T \\ 0, & |\varepsilon(\theta, x, Y)| > T \end{cases}$$

где $\varepsilon(\theta, x)$ – минимальное расстояние от точки x до точек множества Y с выбранной гипотезой θ .

- c. Если мы рассмотрели только первую гипотезу, то сохраняем её в качестве лучшей θ_0 . Иначе проверяем, если текущая гипотеза лучше θ_0 и если лучше, то сохраняем её в качестве лучшей. ($i = 0 \vee (R(\theta_i) > R(\theta_0)) \Rightarrow \theta_0 = \theta_i$)
3. После завершения всех N итераций, θ_0 есть лучшая гипотеза. В нашем случае это матрица преобразования перспективы, которая преобразует первое изображение в систему координат второго изображения.

Вероятность выбора хотя бы одной выборки без точки выброса с помощью метода *RANSAC* может быть оценено так:

$$p = 1 - (1 - N(1 - e)^s)^N,$$

где p – вероятность получения хорошей выборки после N итераций, N – количество итераций, s – число точек в выборке, e – соотношение выбросов (частота выбросов).

Поскольку после оценки хотя бы одной гипотезы мы можем оценить соотношение выбросов, это позволяет нам оценить необходимое количество итераций, основываясь на лучшей на данный момент гипотезе:

$$N = \frac{\log(1 - p)}{\log(1 - (1 - e)^s)}$$

Модифицированный метод *RANSAC* который использует М-оценки для вычисления гипотез, называется *M – SAC*. В его случае оценка каждой точки $p(\theta, x, Y)$ зависит от минимального расстояния от точки x до точек множества Y с гипотезой θ :

$$R(\theta) = \sum_{x \in X} p(\theta, x, Y)$$

$$p(\theta, x, Y) = \begin{cases} \varepsilon^2(\theta, x, Y), & |\varepsilon(\theta, x, Y)| \leq T \\ T^2, & |\varepsilon(\theta, x, Y)| > T \end{cases}$$

Ход выполнения работы

1. Поиск характеристических точек

Исходные изображения:



Figure 7: изображение №1.



Figure 8: изображение №2.



Figure 9: изображение №3.

Будем производить поиск характеристических точек методами *SIFT* и *ORB* последовательно для каждого изображения.

Listing 1. Поиск характеристических точек методом SIFT, используя OpenCV и Python.

```
import cv2 as cv

# Reading image
img_path = "../images/"
img_name = "lakes.jpg"
img = cv.imread(img_path + img_name, cv.IMREAD_COLOR)

# Create SIFT instance
sift = cv.SIFT_create(nfeatures=None)

# Detecting SIFT feature points
img_fp = sift.detect(img)

# Displaying SIFT feature points in green color with scale and orientation
img_out = cv.drawKeypoints(img, img_fp, None, color = (0, 255, 0) ,
flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv.imshow("SIFT detector", img_out)
```

```
cv.waitKey(0)
cv.destroyAllWindows()
```

Listing 2. Поиск характеристических точек методом ORB, используя OpenCV и Python.

```
import cv2 as cv

# Reading image
img_path = "../images/"
img_name = "lakes.jpg"
img = cv.imread(img_path + img_name, cv.IMREAD_COLOR)

# Create ORB instance
orb = cv.ORB_create(nfeatures=None)

# Detecting ORB feature points
img_fp = orb.detect(img)

# Displaying SIFT feature points in green color with scale and orientation
img_out = cv.drawKeypoints(img, img_fp, None, color = (0, 255, 0) ,
flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv.imshow("ORB detector", img_out)
cv.waitKey(0)
cv.destroyAllWindows()
```

1.1. Изображение №1.



Figure 10: характеристические точки с масштабом и ориентацией, найденные методом SIFT.



Figure 11: характеристические точки с масштабом и ориентацией, найденные методом ORB.

Отобразим лучшие 100 точек каждого метода для наглядности.



Figure 12: первые 100 характеристических точек с масштабом и ориентацией, найденные методом SIFT.



Figure 13: первые 100 характеристических точек с масштабом и ориентацией, найденные методом ORB.

1.2. Изображение №2.

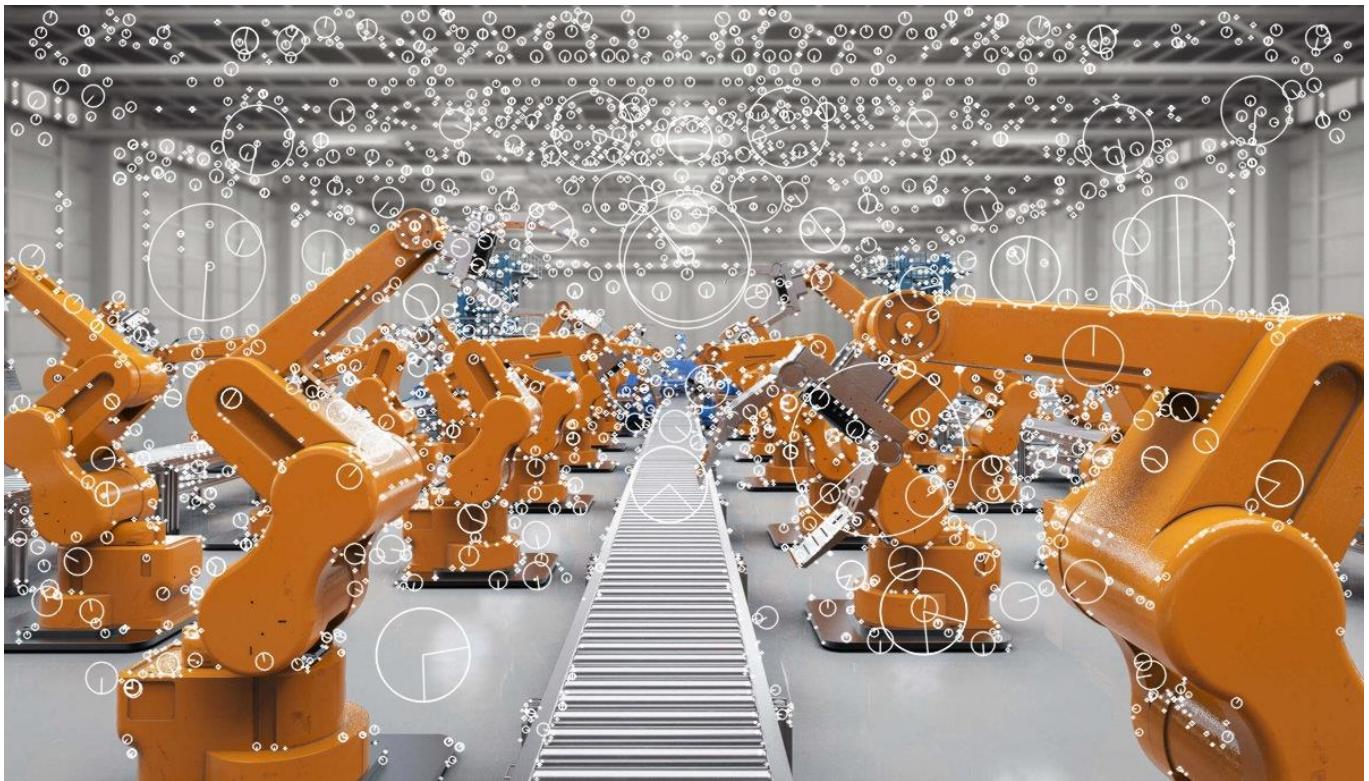


Figure 14: характеристические точки с масштабом и ориентацией, найденные методом SIFT.

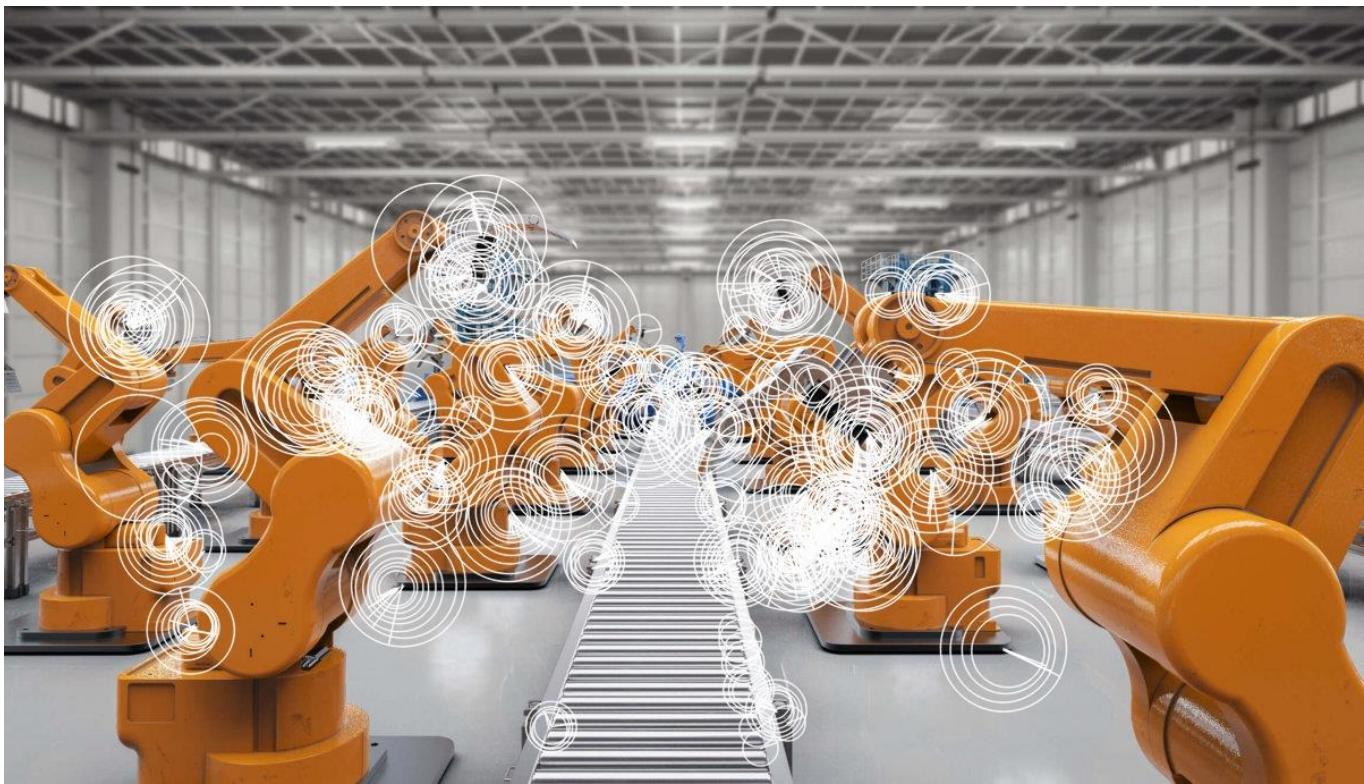


Figure 15: характеристические точки с масштабом и ориентацией, найденные методом ORB.

Отобразим 50 лучших точек.



Figure 16: первые 100 характеристических точки с масштабом и ориентацией, найденные методом SIFT.



Figure 17: первые 100 характеристических точки с масштабом и ориентацией, найденные методом ORB.

1.3. Изображение №3.

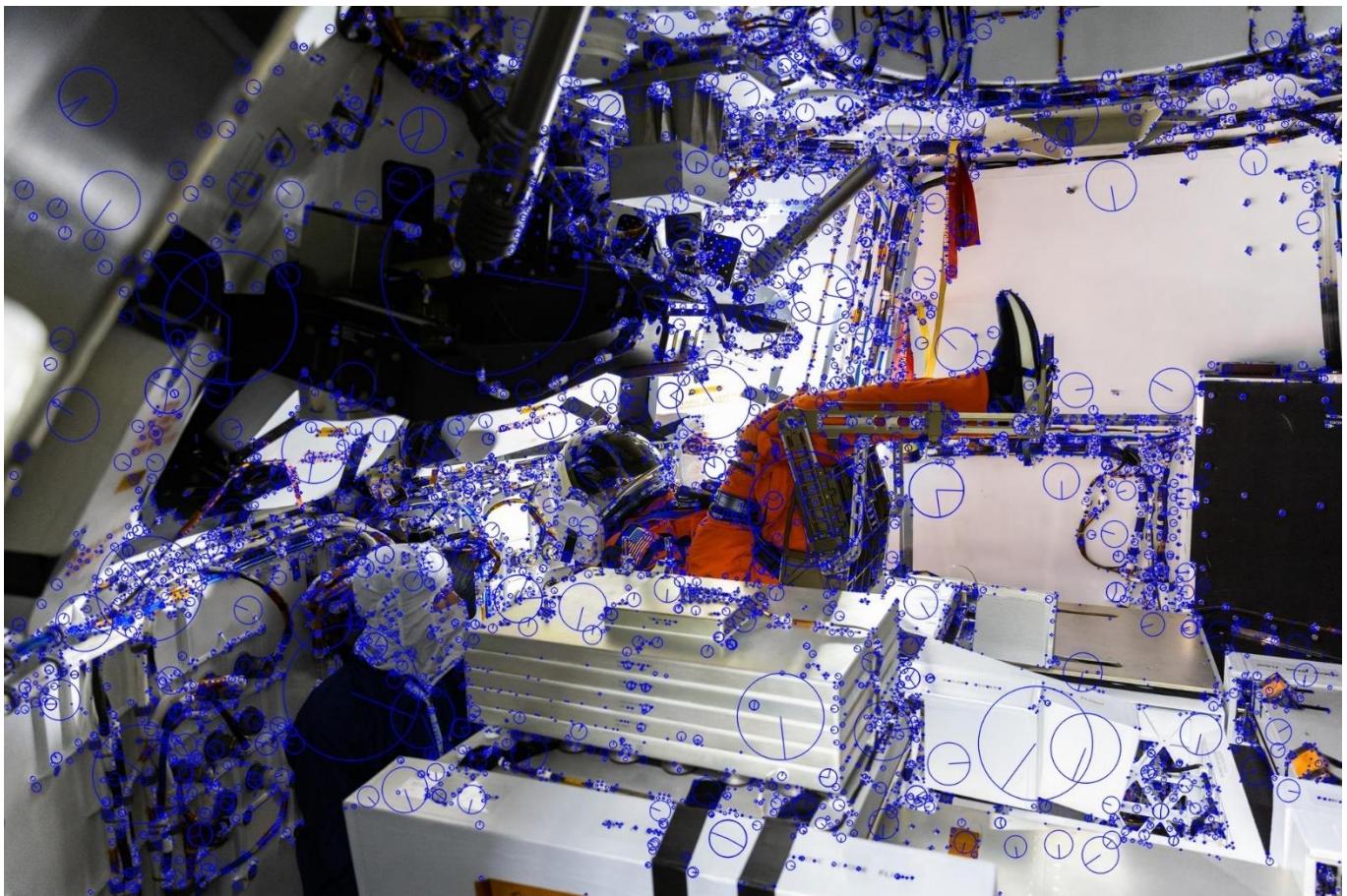


Figure 18: характеристические точки с масштабом и ориентацией, найденные методом SIFT.

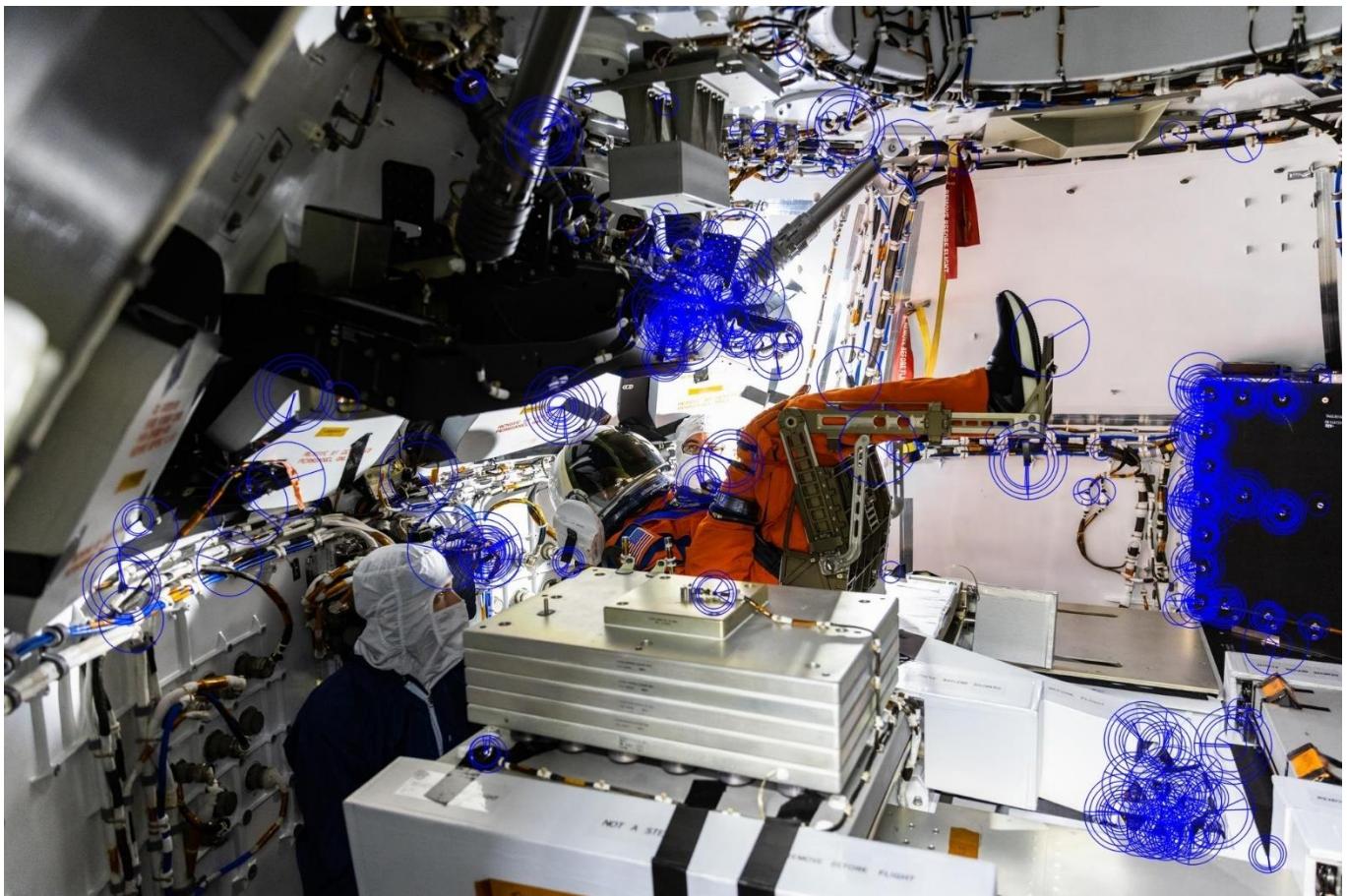


Figure 19: характеристические точки с масштабом и ориентацией, найденные методом ORB.

Отобразим первые 100 характеристических точек.

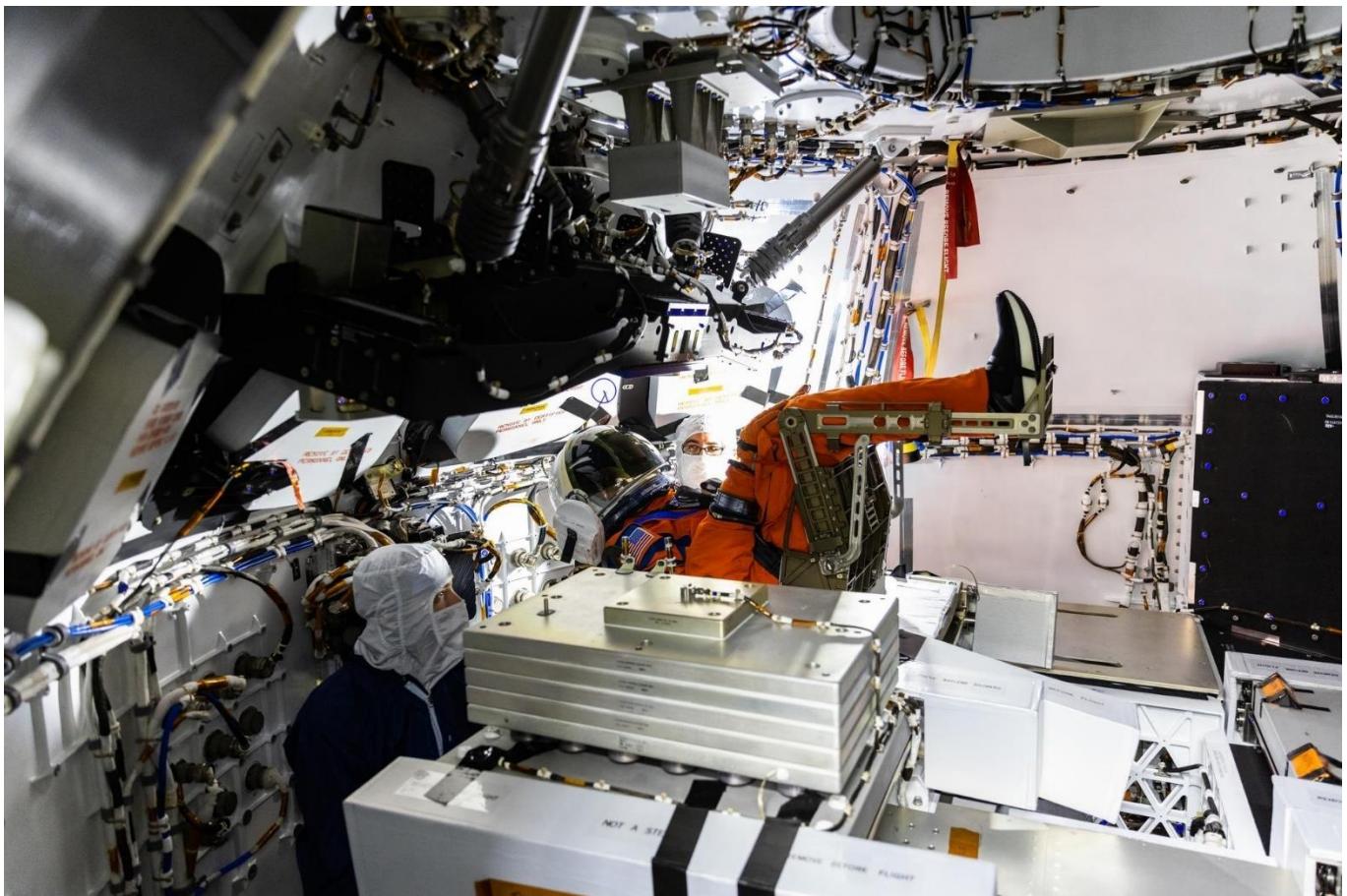


Figure 20: первые 100 характеристических точки с масштабом и ориентацией, найденные методом SIFT.

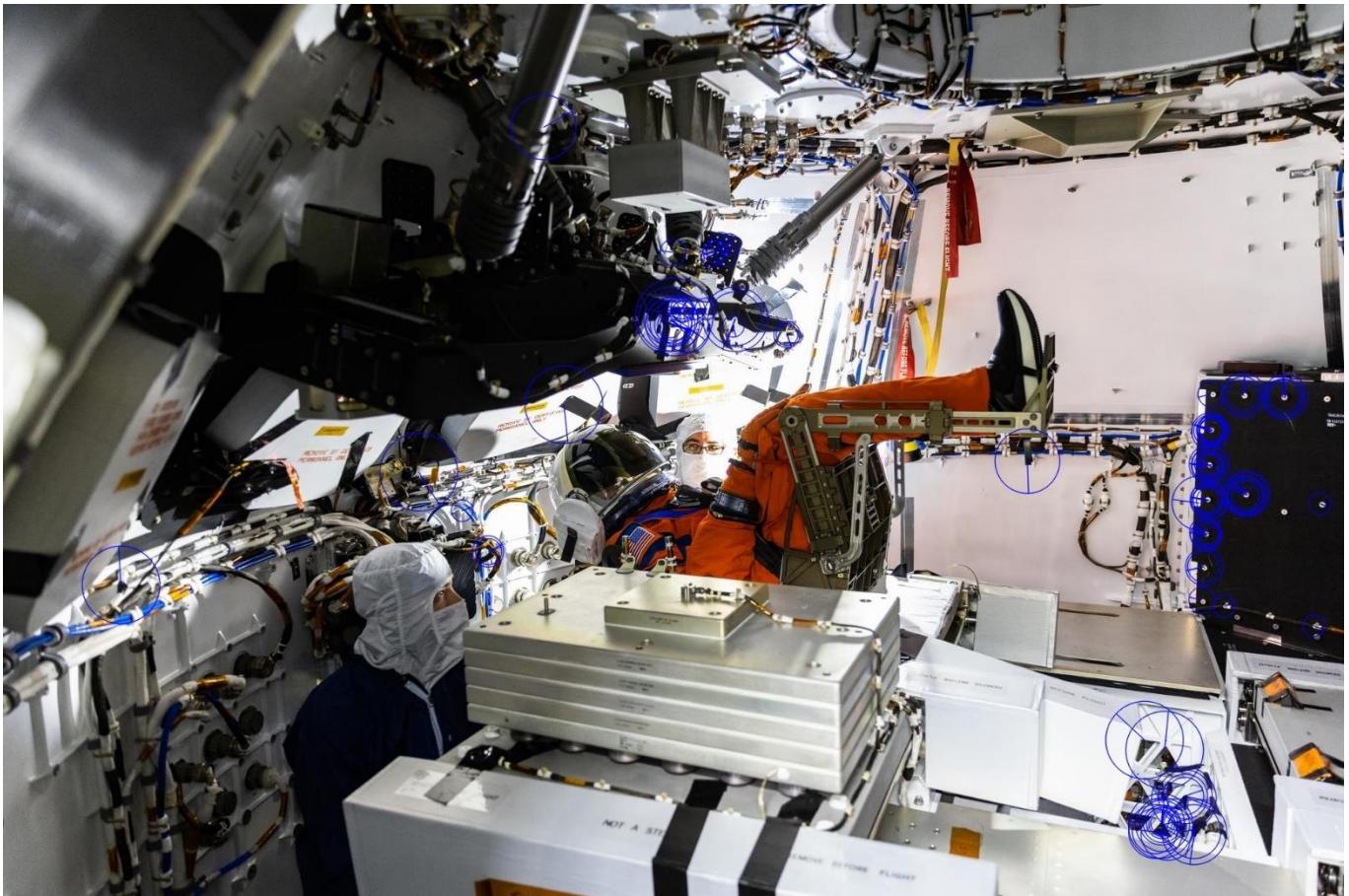


Figure 21: первые 100 характеристических точек с масштабом и ориентацией, найденные методом *ORB*.

1.4. Сравнение методов

Масштаб характеристических точек, найденных методом *ORB* в среднем намного больше, чем у точек, найденных методом *SIFT*.

Также видно, что точки *SIFT* более равномерно распределены по изображению, чего не скажешь о характеристических точкам найденных методом *ORB*, они почти не присутствуют в монотонных областях.

Однако, время поиска характеристических точек методом *SIFT* при работе с большими изображениями (размер изображения №3: 1920x1280) почти в 20 раз больше (*0.733 seconds*), чем при поиске методом *ORB* (*0.0495 seconds*). Причем, время работы этого алгоритма не зависит от параметра *nfeatures*. Исходя из этого можно сделать следующий вывод, что использование метода *ORB* предпочтительно при работе с *real – time* приложениями.

2. Сопоставление характеристических точек

Выбранные пары изображений:



Figure 22: сцена №1.



Figure 23: объект №1.



Figure 24: сцена №2.

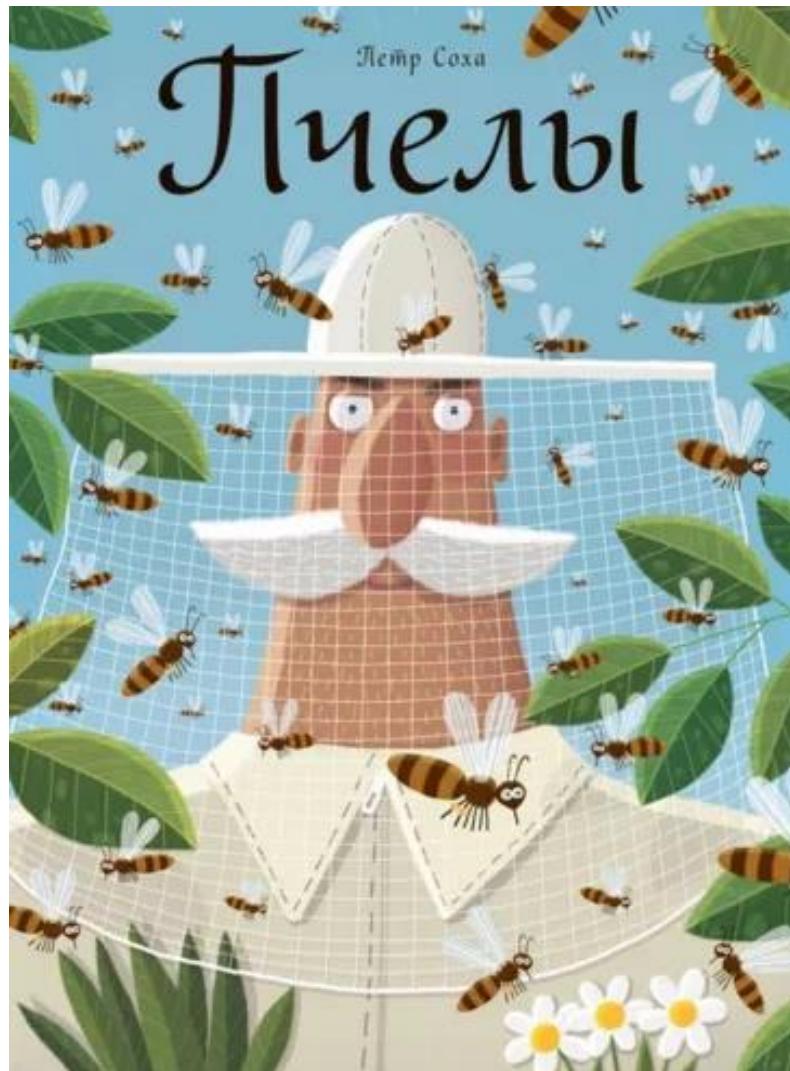


Figure 25: объект №2.

2.1. Сопоставление точек первой пары изображений

В начале извлечем характеристические точки из обоих изображений и опишем их дескриптором *SIFT*.

Listing 2. Чтение изображений, извлечение характеристических точек и расчет дескрипторов методом SIFT, используя OpenCV на Python.

```
import cv2 as cv
import numpy as np

# Reading images
img_path = "../images/"
obj_img_name = "spb_object.png"
scene_img_name = "spb_scene.jpg"
obj_img = cv.imread(img_path + obj_img_name, cv.IMREAD_COLOR)
scene_img = cv.imread(img_path + scene_img_name, cv.IMREAD_COLOR)
```

```

# Detect feature points and compute descriptors
sift = cv.SIFT_create()
obj_fp, obj_descr = sift.detectAndCompute(obj_img, None)
scene_fp, scene_descr = sift.detectAndCompute(scene_img, None)

# Displaying SIFT feature points in green color with scale and orientation
obj_out = cv.drawKeypoints(obj_img, obj_fp, None, color = (0, 255, 0) ,
flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
scene_out = cv.drawKeypoints(scene_img, scene_fp, None, color = (0, 255, 0) ,
flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv.imshow("Object features", obj_out)
cv.imshow("Scene features", scene_out)
cv.waitKey(0)
cv.destroyAllWindows()

```

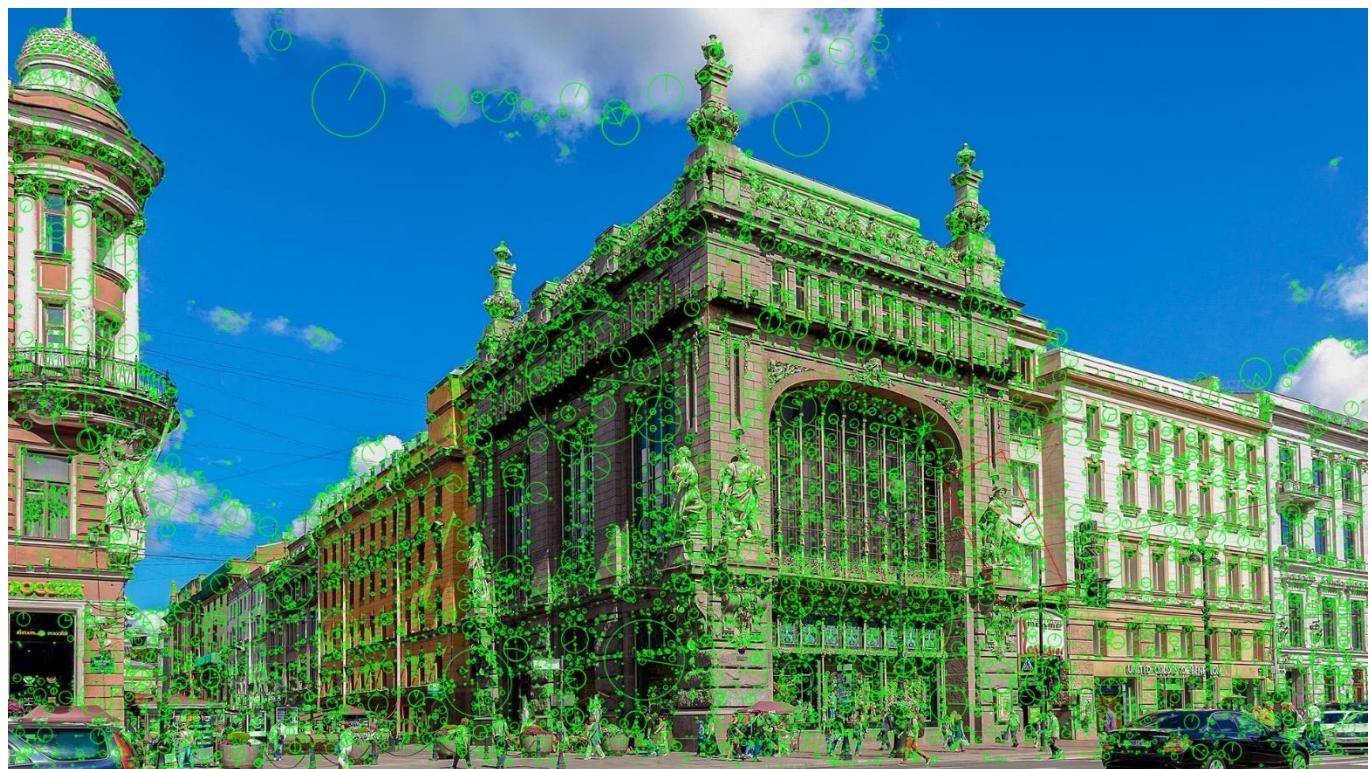


Figure 26: характеристические точки, найденные методом SIFT на сцене №1.



Figure 27: характеристические точки, найденные на объекте №1 методом SIFT.

Теперь создадим объект, который будет сопоставлять характеристические точки методом перебора. Далее используя метод ближайших соседей ($k = 2$) будем находить только сильные совпадения. Затем выведем 50 лучших совпадений.

Listing 3. Сопоставление точек методом перебора и kNN, используя OpenCV и Python.

```
# Creating brute force descriptor matcher
matcher = cv.BFMatcher(crossCheck=False)

# Finding single best match for two sets of descriptors
matches = matcher.match(obj_descr, scene_descr)
```

```

# Finding k-nearest best match for two sets of descriptors and filtering them
# Find kNN matches with k = 2
matches = matcher.knnMatch(obj_descr, scene_descr, k = 2)
# Select good matches
knn_ratio = 0.75
good = []
for m in matches:
    if len(m) > 1:
        if m[0].distance < knn_ratio * m[1].distance:
            good.append(m[0])
matches = good

# Displaying top 50 matches
num_matches = 50
matches = sorted(matches, key = lambda x:x.distance)
img_match = cv.drawMatches(obj_img, obj_fp, scene_img, scene_fp,
matches[:num_matches], None, flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS,
matchColor=(0, 255, 0))
cv.imshow("Matches", img_match)
cv.waitKey(0)
cv.destroyAllWindows()

```

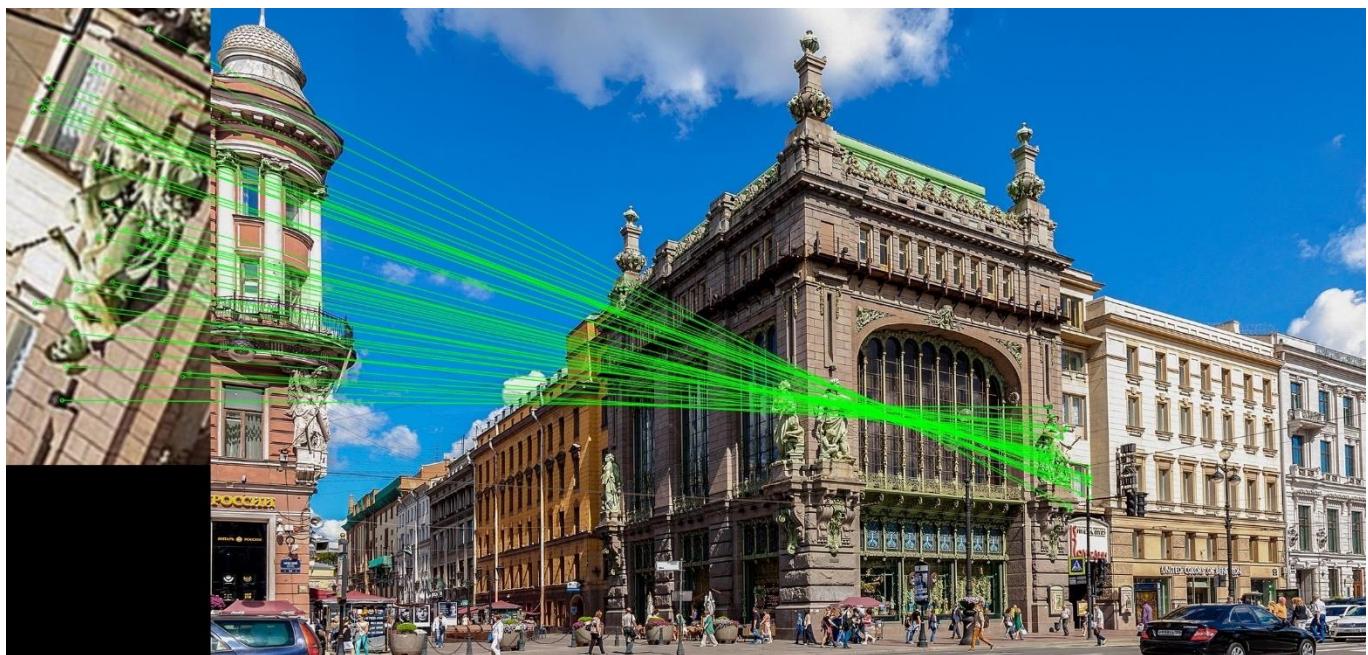


Figure 28: 50 лучших сопоставлений характеристических точек, используя SIFT, метод перебора и kNN.

Теперь применим алгоритм *RANSAC* для расчета матрицы преобразования объекта к изображению сцены. Затем отобразим расположение объекта и правильные сопоставления характеристических точек.

Listing 4. Алгоритм RANSAC и его визуализация, используя OpenCV и Python.

```

# Executing RANSAC to calculate the transformation matrix
MIN_MATCH_COUNT = 10

```

```

if len(matches) < MIN_MATCH_COUNT:
    print("Not enough matches.")

# Create arrays of point coordinates
obj_pts = np.float32([obj_fp[m.queryIdx].pt for m in matches]).reshape(-1, 1, 2)
scene_pts = np.float32([scene_fp[m.trainIdx].pt for m in matches]).reshape(-1, 1, 2)

# Run RANSAC method
M, mask = cv.findHomography(obj_pts, scene_pts, cv.RANSAC, 5)
mask = mask.ravel().tolist()

# Displaying the location of the first image on the second one
# Image corners
h, w = obj_img.shape[:2]
obj_box = np.float32([[0, 0], [0, h - 1], [w - 1, h - 1], [w - 1, 0]]).reshape(-1,
1, 2)
obj_to_scene_box = cv.perspectiveTransform(obj_box, M)
# Draw a red box on the scene image
img_res = cv.polyline(scene_img, [np.int32(obj_to_scene_box)], True, (0, 0, 255),
1, cv.LINE_AA)
cv.imshow("Search result", img_res)
cv.waitKey(0)
cv.destroyAllWindows()

# Displaying inlier matches
img_trans = cv.drawMatches(obj_img, obj_fp, scene_img, scene_fp, matches, None,
matchesMask=mask, flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS, matchColor=(0,
255, 0))
cv.imshow("Transformation", img_trans)
cv.waitKey(0)
cv.destroyAllWindows()

```



Figure 29: расположение объекта №1 на сцене №1, вычисленное с помощью матрицы преобразования (найденной с помощью RANSAC).

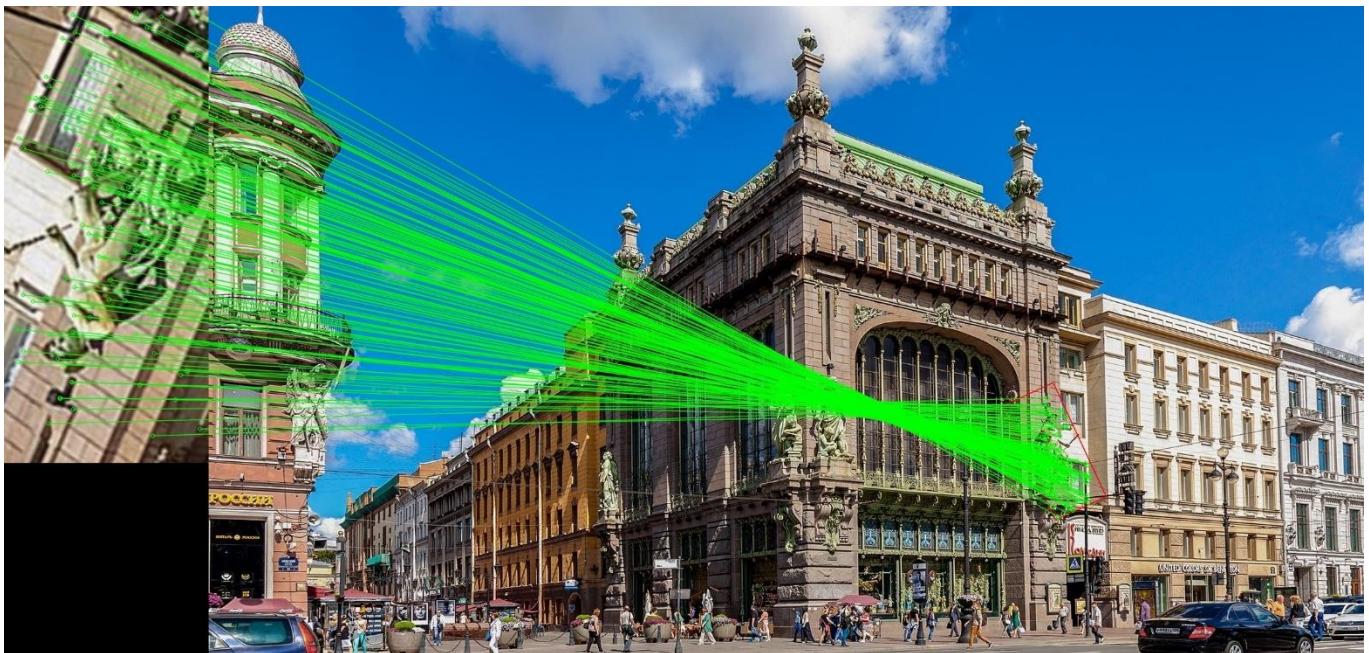


Figure 30: inliers характеристические точки сцены №1 и объекта №1.

Как видно, данный алгоритм успешно справился с задачей поиска объекта на изображении.

Теперь выполним такую же задачу, но с другим дескриптором и детектором. Будем использовать детектор *ORB* и дескриптор *BRIEF*.

Listing 5. Вычисление характеристических точек и их сигнатур методом ORB, используя OpenCV и Python.

```
# Detect feature points and compute descriptors
orb = cv.ORB_create()
obj_fp, obj_descr = orb.detectAndCompute(obj_img, None)
scene_fp, scene_descr = orb.detectAndCompute(scene_img, None)
```

Все остальные шаги алгоритма остаются те же. Предоставим результаты работы.



Figure 31: характеристические точки с масштабом и ориентацией сцены №1 вычисленные детектором ORB.



Figure 32: характеристические точки с масштабом и ориентацией объекта №1 вычисленные детектором ORB.

Для вычисления расстояние между характеристическими точками используем норму Хэмминга, так как дескриптор имеет вид бинарной строки.

Listing 6. Создание дескриптора с нормой Хэмминга, используя OpenCV и Python.

```
# Creating brute force descriptor matcher with Hamming distance
matcher = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=False)
```



Figure 33: лучшие сопоставления характеристических точек, используя детектор ORB.

Как видим, количество точек меньше 10 (равно 5), а следовательно мы не сможем применить метод *RANSAC* для нахождения матрицы преобразования изображения объекта к изображению сцены.

2.2. Сопоставление характеристических точек второй пары изображений

Будем использовать тот же алгоритм что и для первой пары изображений.

Сначала применим метод *SIFT*, затем метод *ORB*.



Figure 34: характеристические точки с масштабом и ориентацией сцены №2, вычисленные детектором SIFT.

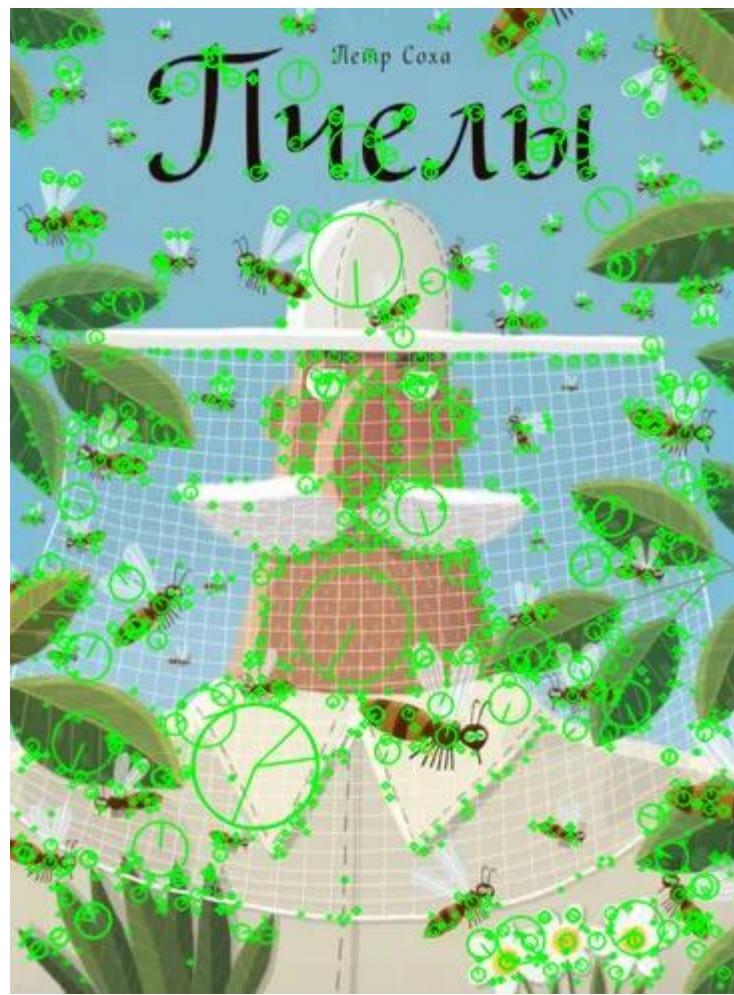


Figure 35: характеристические точки с масштабом и ориентацией объекта №2, вычисленные детектором SIFT.



Figure 36: 50 лучших сопоставлений характеристических точек сцены №2 и объекта №2, используя SIFT, метод перебора и kNN.



Figure 37: расположение объекта №2 на сцене №2, вычисленное с помощью матрицы преобразования (найденной с помощью RANSAC).



Figure 38: *inliers* объекта №2 и сцены №2, используя детектор и дескриптор SIFT.

Удалось найти книгу на сцене, несмотря на различную (хоть и немного *viewpoint*), а также перекрытие объекта.

Теперь применим тот же алгоритм, но с использованием детектора *ORB*.



Figure 39: характеристические точки с масштабом и ориентацией сцены №2, вычисленные детектором ORB.

Как видно, на интересующем нас объекте не было найдено ни одной характеристической точки. Поэтому поступим следующим образом: найдем характеристические точки детектором *SIFT*, а вычислим для них сигнатуру с помощью дескриптора *BRIEF*, являющегося частью *ORB*.

Listing 7. Детектор SIFT и дескриптор BRIEF, используя OpenCV и Python.

```
# Detect feature points and compute descriptors
sift = cv.SIFT_create()
brief = cv.xfeatures2d.BriefDescriptorExtractor_create()

obj_fp = sift.detect(obj_img, None)
scene_fp = sift.detect(scene_img, None)

obj_fp, obj_descr = brief.compute(obj_img, obj_fp)
scene_fp, scene_descr = brief.compute(scene_img, scene_fp)
```

Соответственно, будем использовать метрику Хэмминга, так как сигнатура наших характеристических точек имеет вид битовой строки.



Figure 40: 50 лучших сопоставлений характеристических точек сцены №2 и объекта №2, используя SIFT + BRIEF, метод перебора и kNN.



Figure 41: расположение объекта №2 на сцене №2, вычисленное с помощью матрицы преобразования (найденной с помощью RANSAC) и комбинации SIFT + BRIEF.



Figure 42: *inliers* объекта №2 и сцены №2, используя детектор SIFT и дескриптор BRIEF.

Как видим, алгоритм с дескриптором *BRIEF* тоже справился с задачей.

2.3. Сравнение *ORB(BRIEF)* и *SIFT*

По сути, *BRIEF* и *ORB* работают быстрее, так как вычисление расстояния Хэмминга является практически молниеносным по сравнению с евклидовым, не говоря уже о хранении данных.

Однако на практике я убедился, что детектор *SIFT* работает существенно лучше, нежели детектор *ORB*, хотя разница во времени и тут не в пользу *SIFT*.

В итоге применение того или иного метода необходимо выбирать исходя из условия задачи, которая стоит перед исследователем. Поэтому необходимо знать и уметь применять каждый алгоритм [рассмотренный тут](#).

3. Склейивание изображений

В данной части работы будет произведено склейивание 3-х изображений в одно панорамное.

Предполагается, что у нас есть информация о порядке изображений.

Склейка будет производиться по следующему алгоритму:

- Для всех изображений найдем их характеристические точки детектором *SIFT* и опишем их дескриптором *SIFT*

- Для двух соседних изображений (две пары соседних изображений) сопоставим их характеристические точки методом перебора
- Применим метод ближайших соседей для сопоставленных характеристических точек и выделим с помощью порогового значения «сильные» соответствия
- Затем используя множество соответствий характеристических точек, применим алгоритм *RANSAC* для вычисления матрицы преобразования M одного изображения к другому. Будем преобразовывать крайние изображения в систему координат среднего
- Далее найдем *bounding boxes* крайних изображений с помощью матрицы M и исходя из их углов, вычислим *paddings*, которые используем для вычисления размеров панорамного изображения
- Применим операцию сдвига наших *bounding boxes*, чтобы они поместились полностью в итоговую панорамную картинку
- Произведем преобразование крайних картинок с помощью матрицы трансляции (вычисленной с помощью *paddings*) и матрицы M
- В конце вычислим маски изображений и соединим их в одно с помощью логических операций

Listing 8. Чтение изображений, определение характеристических точек и вычисление их сигнатур методом SIFT, используя OpenCV на Python.

```
import cv2 as cv
import numpy as np

# Reading images
img_path = "../images/"
img_1_name = "room_1.jpg"
img_2_name = "room_2.jpg"
img_3_name = "room_3.jpg"
img_1 = cv.imread(img_path + img_1_name, cv.IMREAD_COLOR)
img_2 = cv.imread(img_path + img_2_name, cv.IMREAD_COLOR)
img_3 = cv.imread(img_path + img_3_name, cv.IMREAD_COLOR)

# Detect feature points and compute descriptors
sift = cv.SIFT_create()

img_1_fp, img_1_des = sift.detectAndCompute(img_1, None)
img_2_fp, img_2_des = sift.detectAndCompute(img_2, None)
img_3_fp, img_3_des = sift.detectAndCompute(img_3, None)

# Displaying SIFT feature points with scale and orientation
img_1_out = cv.drawKeypoints(img_1, img_1_fp, None, color = (0, 255, 0) ,
flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
img_2_out = cv.drawKeypoints(img_2, img_2_fp, None, color = (0, 255, 0) ,
flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
img_3_out = cv.drawKeypoints(img_3, img_3_fp, None, color = (0, 255, 0) ,
flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

img_out = np.concatenate((img_1_out, img_2_out, img_3_out), axis=1)

cv.imwrite(img_path + "room_features.jpg", img_out)
```

```

cv.imshow("Features", img_out)
cv.waitKey(0)
cv.destroyAllWindows()

```



Figure 43: характеристические точки 3-х изображений, найденные методом SIFT.

Listing 9. Нахождение совпадений характеристических точек методом перебора, выбор «сильных» методом kNN и вывод 50 лучших сопоставлений, используя OpenCV и Python.

```

# Creating brute force descriptor matcher
matcher = cv.BFMatcher(crossCheck=False)

def find_best_matches(query_des, train_des, k, knn_ratio):
    # Find kNN matches with k = 2
    matches = matcher.knnMatch(query_des, train_des, k=k)
    # Select good matches
    good = []
    for m in matches:
        if len(m) > 1:
            if m[0].distance < knn_ratio * m[1].distance:
                good.append(m[0])
    return good

# Finding k-nearest best match for 1-2 and 3-2 descriptors of images and filtering them
matches_1_2 = find_best_matches(img_1_des, img_2_des, 2, 0.75)
matches_3_2 = find_best_matches(img_3_des, img_2_des, 2, 0.75)

# Displaying top 50 matches
num_matches = 50
matches_1_2 = sorted(matches_1_2, key = lambda x:x.distance)
matches_3_2 = sorted(matches_3_2, key = lambda x:x.distance)

```

```

img_match_1_2 = cv.drawMatches(img_1, img_1_fp, img_2, img_2_fp,
matches_1_2[:num_matches], None, flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS,
matchColor=(255, 0, 0))
img_match_3_2 = cv.drawMatches(img_3, img_3_fp, img_2, img_2_fp,
matches_3_2[:num_matches], None, flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS,
matchColor=(255, 0, 0))

cv.imwrite(img_path + "matches_1_2.jpg", img_match_1_2)
cv.imwrite(img_path + "matches_3_2.jpg", img_match_3_2)
cv.imshow("Matches", img_match_1_2)
cv.imshow("Matches", img_match_3_2)
cv.waitKey(0)
cv.destroyAllWindows()

```

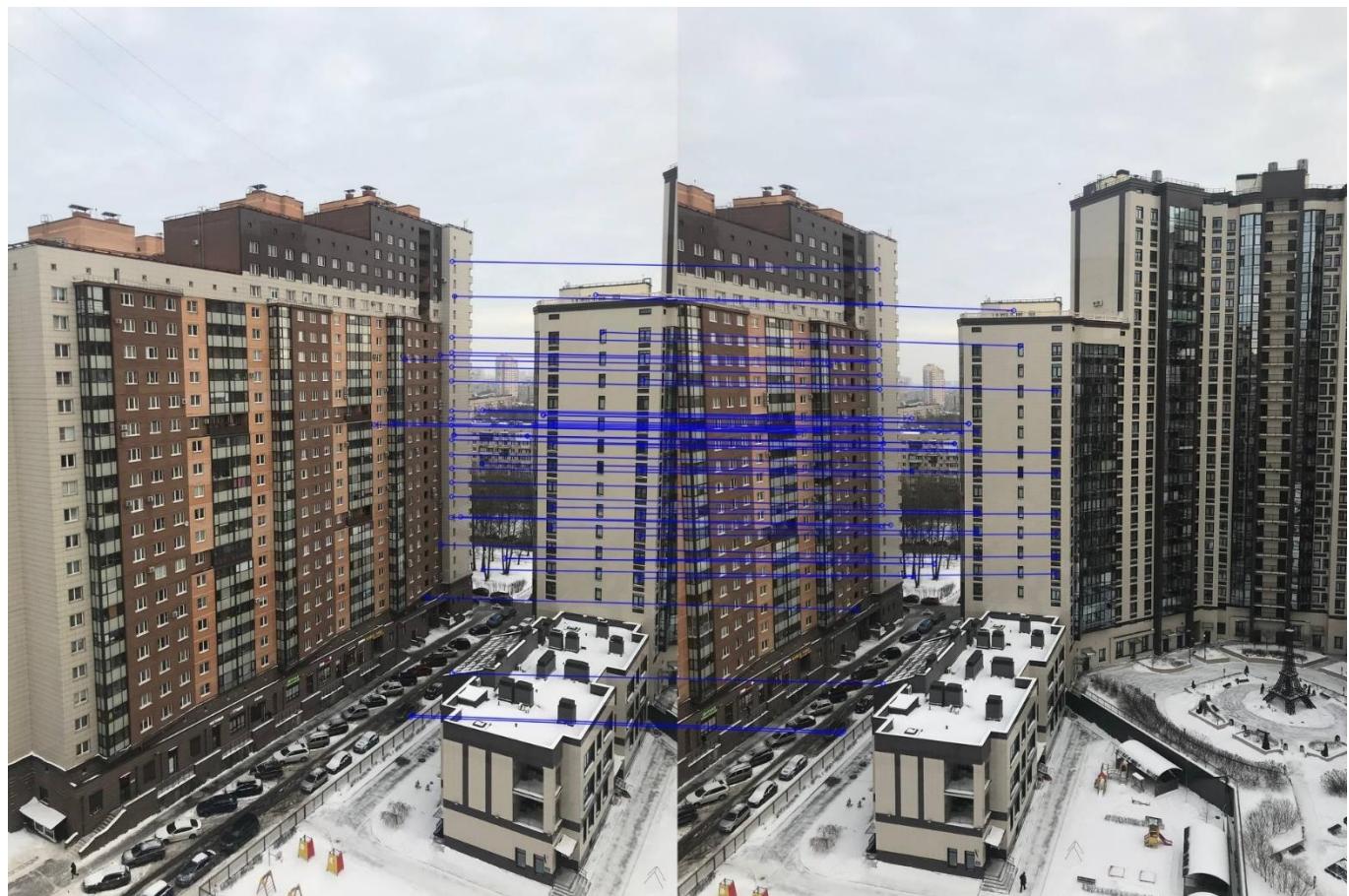


Figure 44: 50 лучших сопоставлений 1 и 2 изображений.

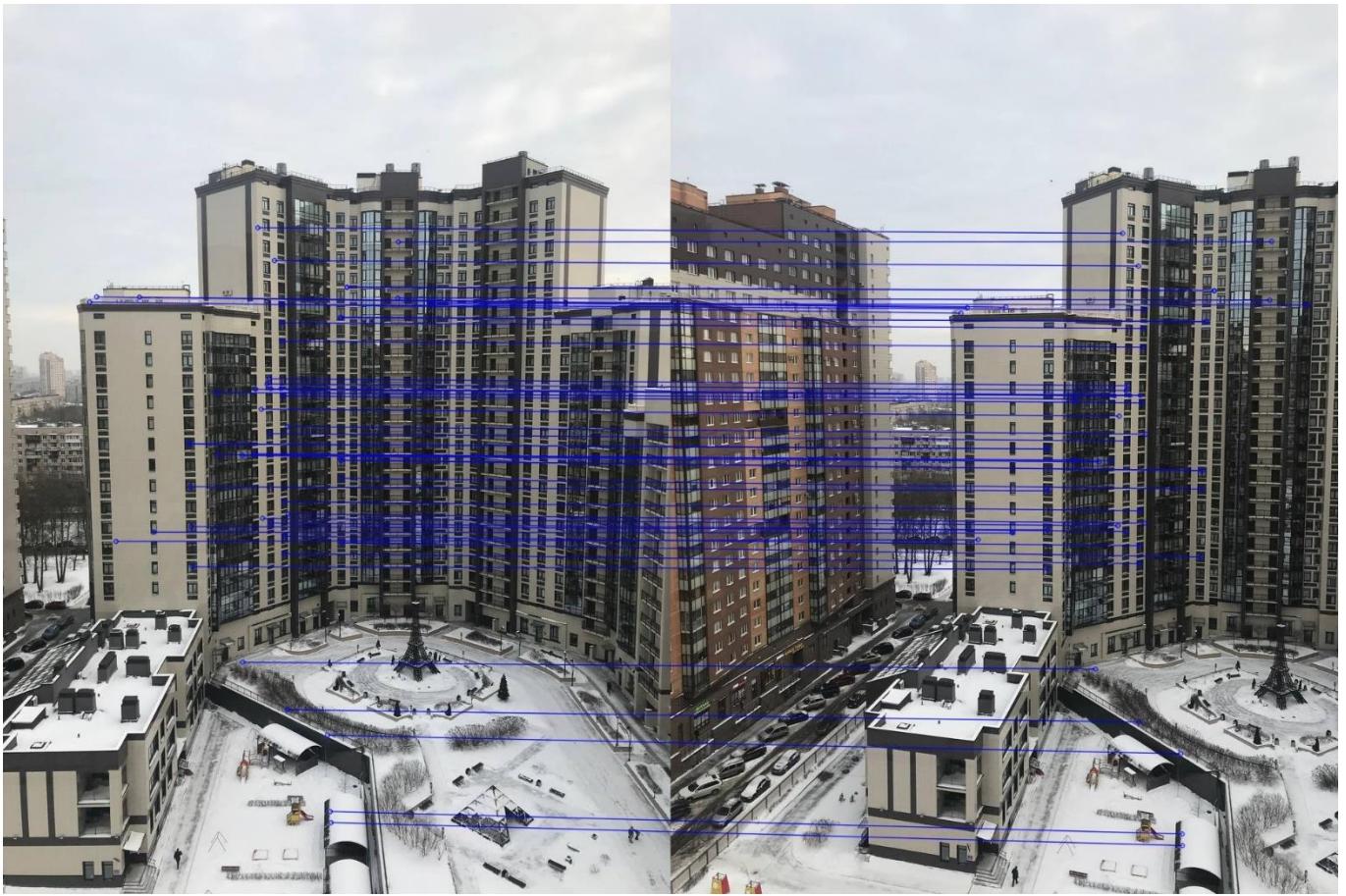


Figure 45: 50 лучших сопоставлений 3 и 2 изображений.

Listing 10. Применение метода RANSAC для поиска матрицы преобразования, а также вывод inliers соотвествий, используя OpenCV и Python.

```
# Executing RANSAC to calculate the transformation matrix
MIN_MATCH_COUNT = 10
if len(matches_1_2) < MIN_MATCH_COUNT or len(matches_3_2) < MIN_MATCH_COUNT:
    print("Not enough matches.")

# Create arrays of point coordinates
img_1_pts = np.float32([img_1_fp[m.queryIdx].pt for m in matches_1_2]).reshape(-1,
1, 2)
img_2_1_pts = np.float32([img_2_fp[m.trainIdx].pt for m in matches_1_2]).reshape(-1,
1, 2)

img_3_pts = np.float32([img_3_fp[m.queryIdx].pt for m in matches_3_2]).reshape(-1,
1, 2)
img_2_3_pts = np.float32([img_2_fp[m.trainIdx].pt for m in matches_3_2]).reshape(-1,
1, 2)

# Run RANSAC method
M_1_2, mask_1_2 = cv.findHomography(img_1_pts, img_2_1_pts, cv.RANSAC, 5)
mask_1_2 = mask_1_2.ravel().tolist()

M_3_2, mask_3_2 = cv.findHomography(img_3_pts, img_2_3_pts, cv.RANSAC, 5)
mask_3_2 = mask_3_2.ravel().tolist()
```

```

# Displaying inlier matches
img_trans_1_2 = cv.drawMatches(img_1, img_1_fp, img_2, img_2_fp, matches_1_2, None,
matchesMask=mask_1_2, flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS,
matchColor=(0, 255, 0))
img_trans_3_2 = cv.drawMatches(img_3, img_3_fp, img_2, img_2_fp, matches_3_2, None,
matchesMask=mask_3_2, flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS,
matchColor=(0, 255, 0))
cv.imwrite(img_path + "img_inliers_1_2.jpg", img_trans_1_2)
cv.imwrite(img_path + "img_inliers_3_2.jpg", img_trans_3_2)
cv.imshow("1 to 2", img_trans_1_2)
cv.imshow("3 to 2", img_trans_3_2)
cv.waitKey(0)
cv.destroyAllWindows()

```

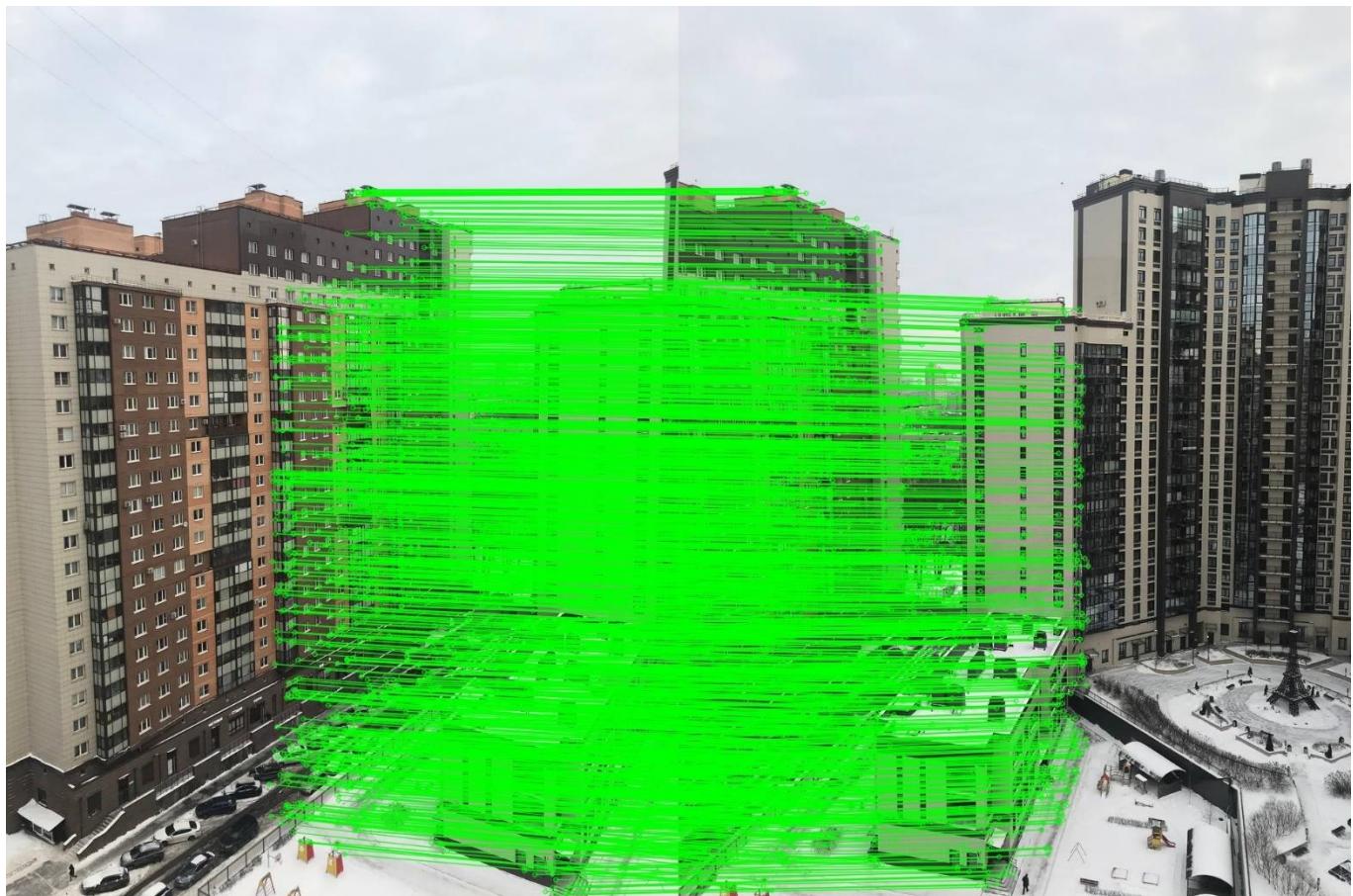


Figure 46: inliers соотвествия 1 к 2 изображений.

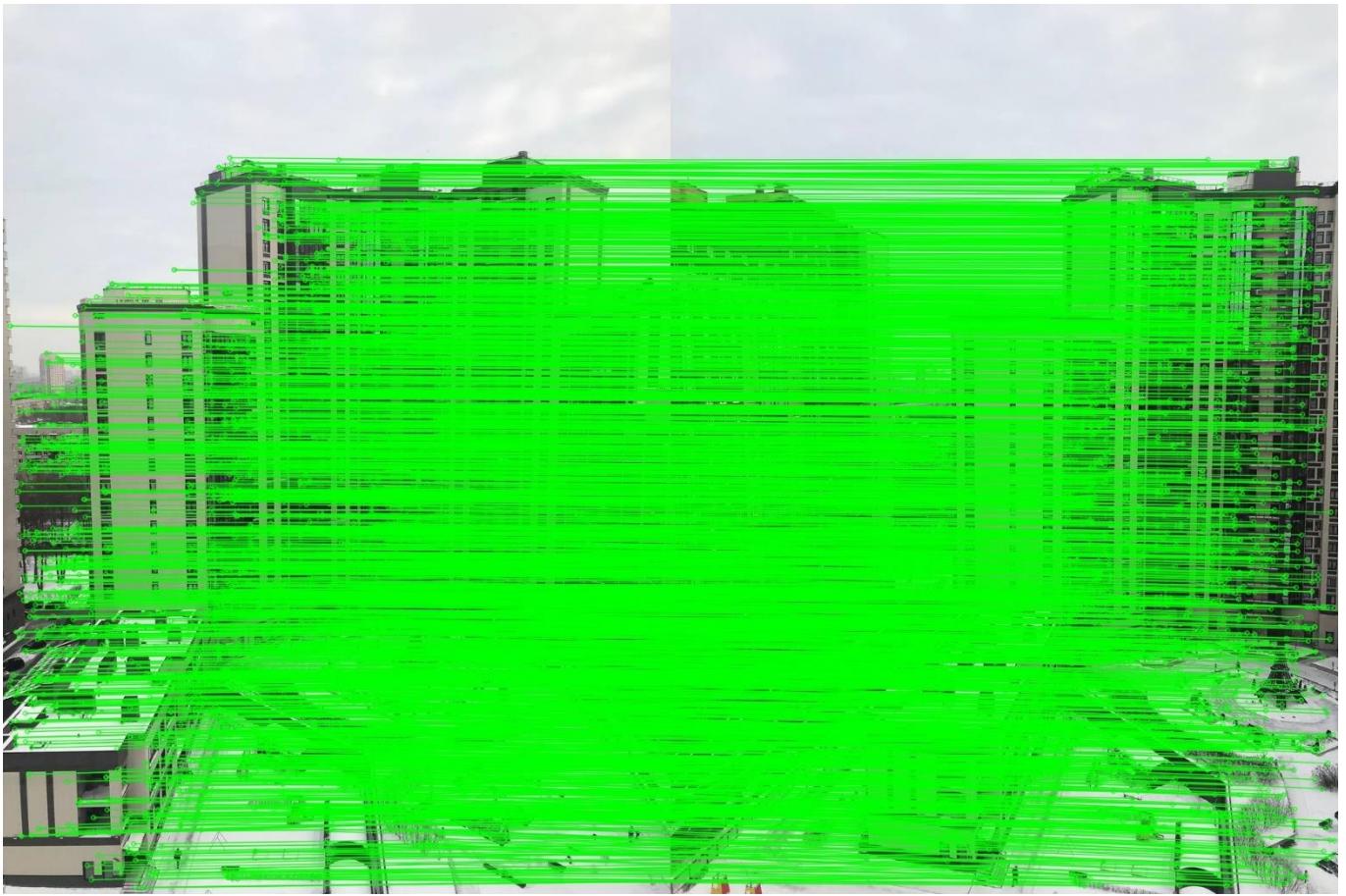


Figure 47: inliers соотвествия 3 к 2 изображений.

Listing 11. Вычисление углов крайних изображений и отступов, также создание итогового шаблона панорамного изображения, используя OpenCV и Python.

```
# Images corners
h_1, w_1 = img_1.shape[:2]
h_3, w_3 = img_3.shape[:2]
img_1_box = np.float32([[0, 0], [0, h_1 - 1], [w_1 - 1, h_1 - 1], [w_1 - 1, 0]]).reshape(-1, 1, 2)
img_3_box = np.float32([[0, 0], [0, h_3 - 1], [w_3 - 1, h_3 - 1], [w_3 - 1, 0]]).reshape(-1, 1, 2)
img_1_to_img_2_box = cv.perspectiveTransform(img_1_box, M_1_2)
img_3_to_img_2_box = cv.perspectiveTransform(img_3_box, M_3_2)

# Compute paddings
top_padding = int(max(-img_1_to_img_2_box[0][0, 1], -img_3_to_img_2_box[3][0, 1], 0))
bot_padding = int(max(img_1_to_img_2_box[1][0, 1] - img_2.shape[0], img_3_to_img_2_box[2][0, 1] - img_2.shape[0], 0))
left_padding = int(max(-img_1_to_img_2_box[0][0, 0], -img_1_to_img_2_box[1][0, 0], 0))
right_padding = int(max(img_3_to_img_2_box[3][0, 0] - img_2.shape[1], img_3_to_img_2_box[2][0, 0] - img_2.shape[1], 0))
```

```
# Create panoramic image
panoramic_img = cv.copyMakeBorder(img_2, top_padding, bot_padding, left_padding,
right_padding, borderType=cv.BORDER_CONSTANT)
cv.imwrite(img_path + "panoramic_img.jpg", panoramic_img)
```

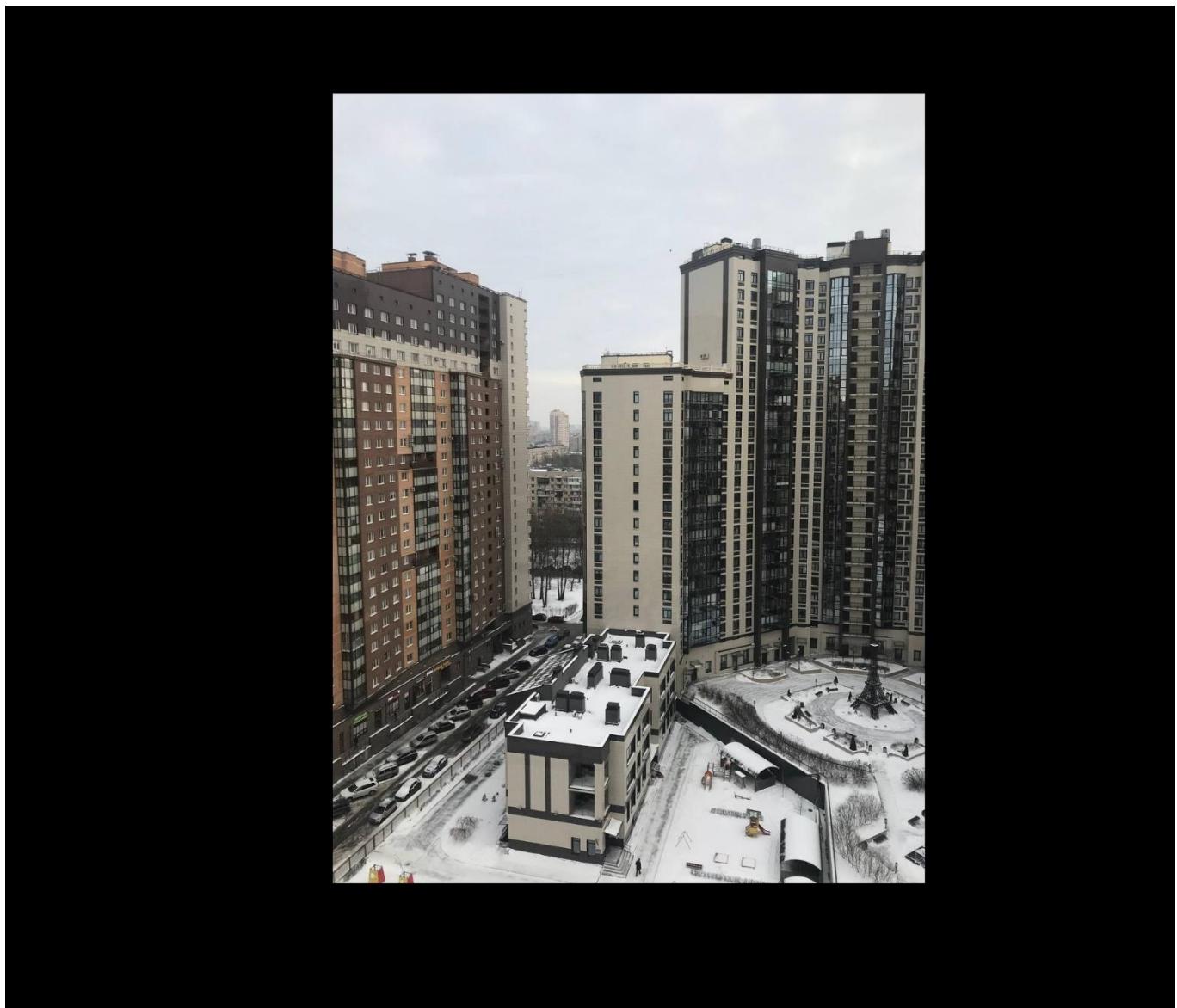


Figure 48: шаблон панорамы.

Listing 12. Отображение границ крайних изображений на панораме.

```
# Shift images boxes
translation_matrix = np.array([[1, 0, left_padding], [0, 1, top_padding], [0, 0,
1]])
img_1_to_img_2_box_shifted = cv.perspectiveTransform(img_1_to_img_2_box,
translation_matrix)
img_3_to_img_2_box_shifted = cv.perspectiveTransform(img_3_to_img_2_box,
translation_matrix)
```

```

# Draw boxes on panoramic image
pano_with_boxes = np.copy(panoramic_img)
pano_with_boxes = cv.polyline(pano_with_boxes,
[img_1_to_img_2_box_shifted], True, (255, 0, 0), 3, cv.LINE_AA)
pano_with_boxes = cv.polyline(pano_with_boxes,
[img_3_to_img_2_box_shifted], True, (255, 0, 0), 3, cv.LINE_AA)
cv.imwrite(img_path + "pano_boxes.jpg", pano_with_boxes)

```

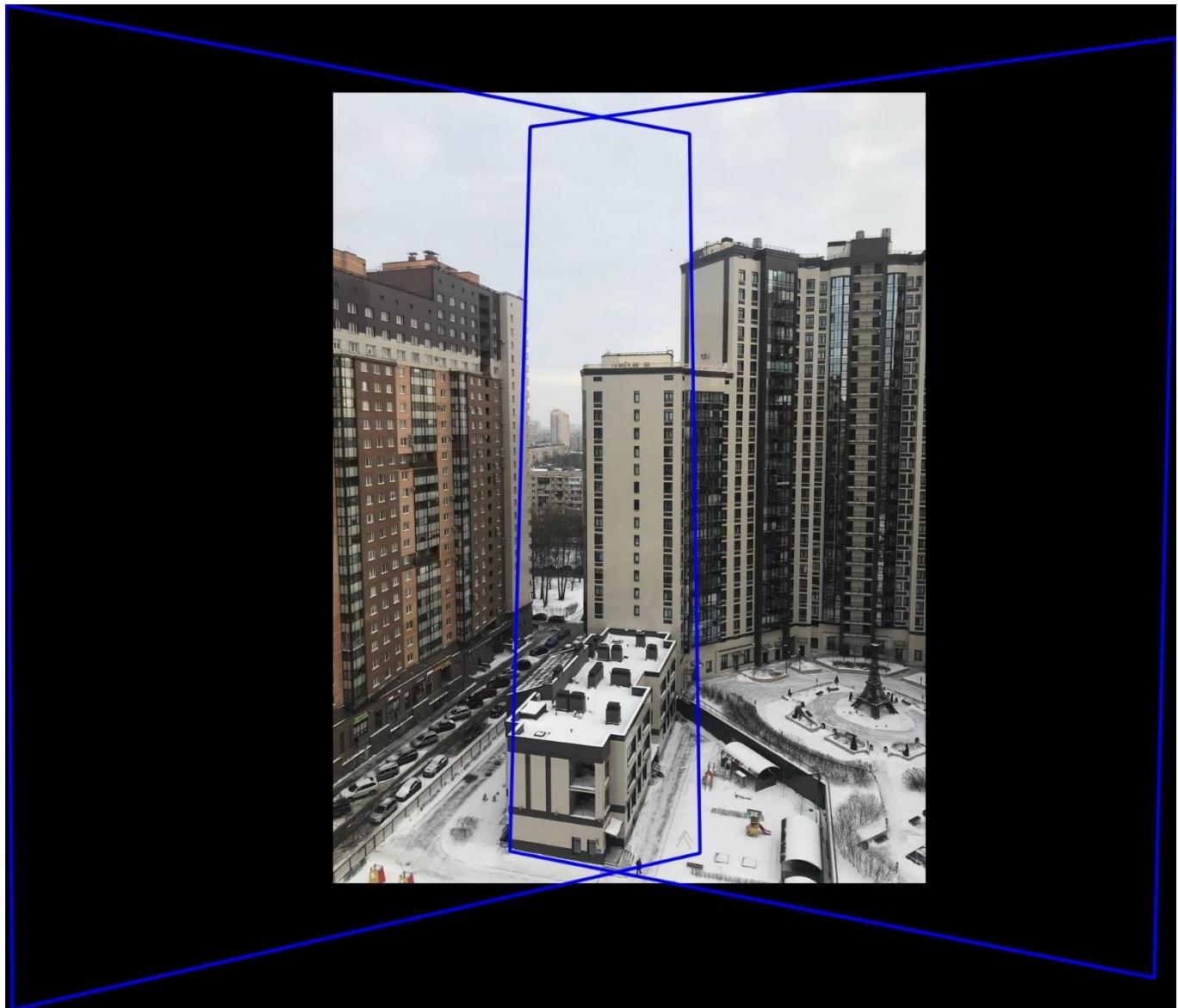


Figure 49: расположение крайних изображений на панораме.

Listing 13. Применение преобразования к крайним изображениям.

```

# Perform warp for images 1 and 3
img_1_warp = cv.warpPerspective(img_1, np.matmul(translation_matrix, M_1_2),
dsize=(panoramic_img.shape[1], panoramic_img.shape[0]))
img_3_warp = cv.warpPerspective(img_3, np.matmul(translation_matrix, M_3_2),
dsize=(panoramic_img.shape[1], panoramic_img.shape[0]))

```

```
cv.imwrite(img_path + "img_1_warp.jpg", img_1_warp)
cv.imwrite(img_path + "img_3_warp.jpg", img_3_warp)
```

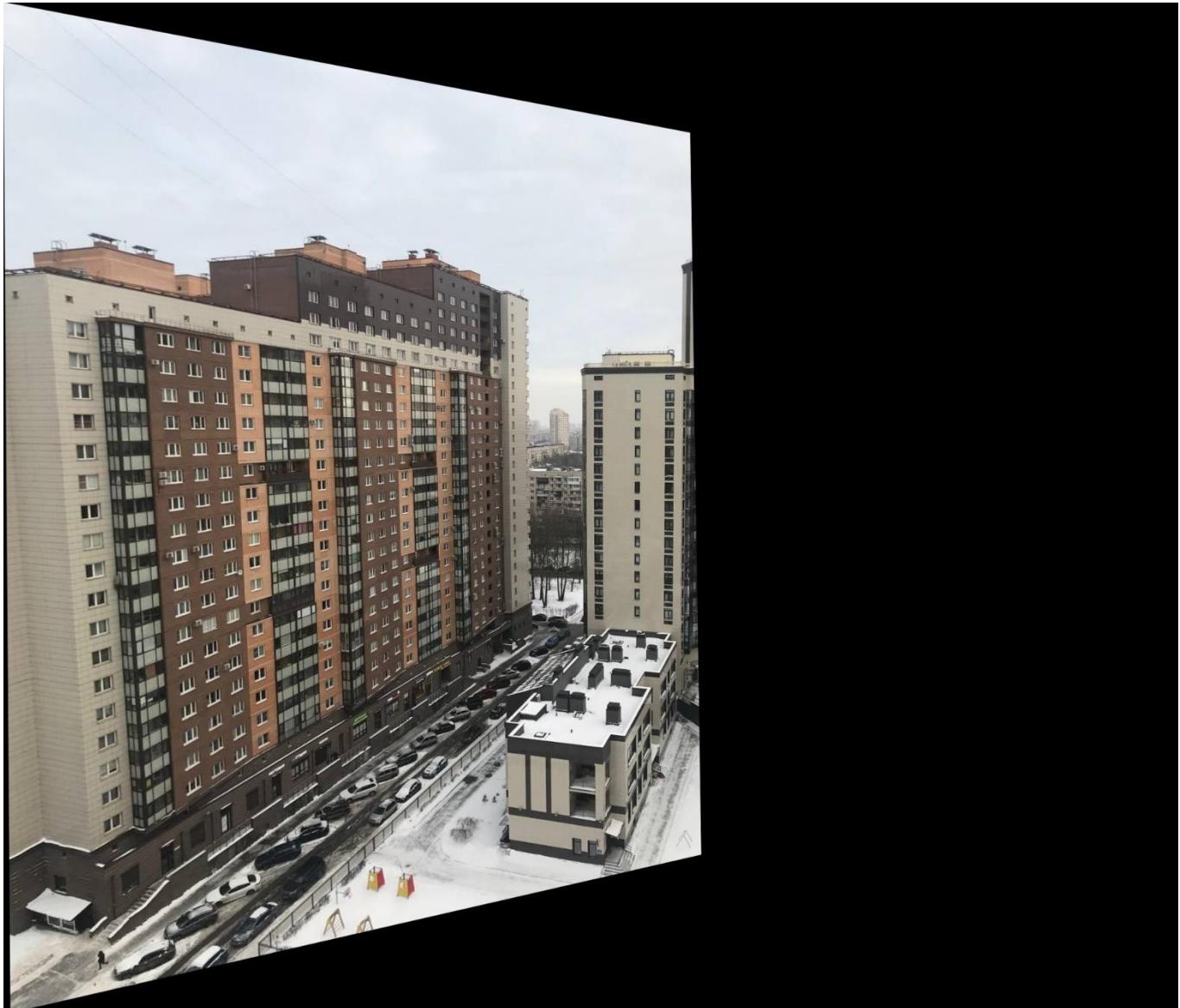


Figure 50: преобразование 1 изображения.



Figure 51: преобразование 2 изображения.

Listing 14. Создание масок изображений и их склеивание.

```
# Create masks for images 1, 2, 3
pan_h, pan_w = panoramic_img.shape[:2]

img_1_mask = np.zeros((pan_h, pan_w), dtype=np.uint8)
img_1_mask = cv.fillConvexPoly(img_1_mask, np.int32(img_1_to_img_2_box_shifted),
255)

img_2_mask = np.zeros((pan_h, pan_w), dtype=np.uint8)
img_2_mask[top_padding:(pan_h - bot_padding), left_padding:(pan_w - right_padding)] =
255

img_3_mask = np.zeros((pan_h, pan_w), dtype=np.uint8)
img_3_mask = cv.fillConvexPoly(img_3_mask, np.int32(img_3_to_img_2_box_shifted),
255)
```

```
# Save masks
cv.imwrite(img_path + "img_1_mask.jpg", img_1_mask)
cv.imwrite(img_path + "img_2_mask.jpg", img_2_mask)
cv.imwrite(img_path + "img_3_mask.jpg", img_3_mask)

# Merge all images into panoramic one

img_1_3_mask = cv.bitwise_xor(img_1_mask, img_3_mask) -
cv.bitwise_and(cv.bitwise_xor(img_1_mask, img_3_mask), img_2_mask)

img_1_warp_cropped = cv.bitwise_and(img_1_warp, img_1_warp, mask=img_1_3_mask)
img_3_warp_cropped = cv.bitwise_and(img_3_warp, img_3_warp, mask=img_1_3_mask)

result = cv.add(cv.add(img_1_warp_cropped, img_3_warp_cropped), panoramic_img)

cv.imwrite(img_path + "result.jpg", result)
```

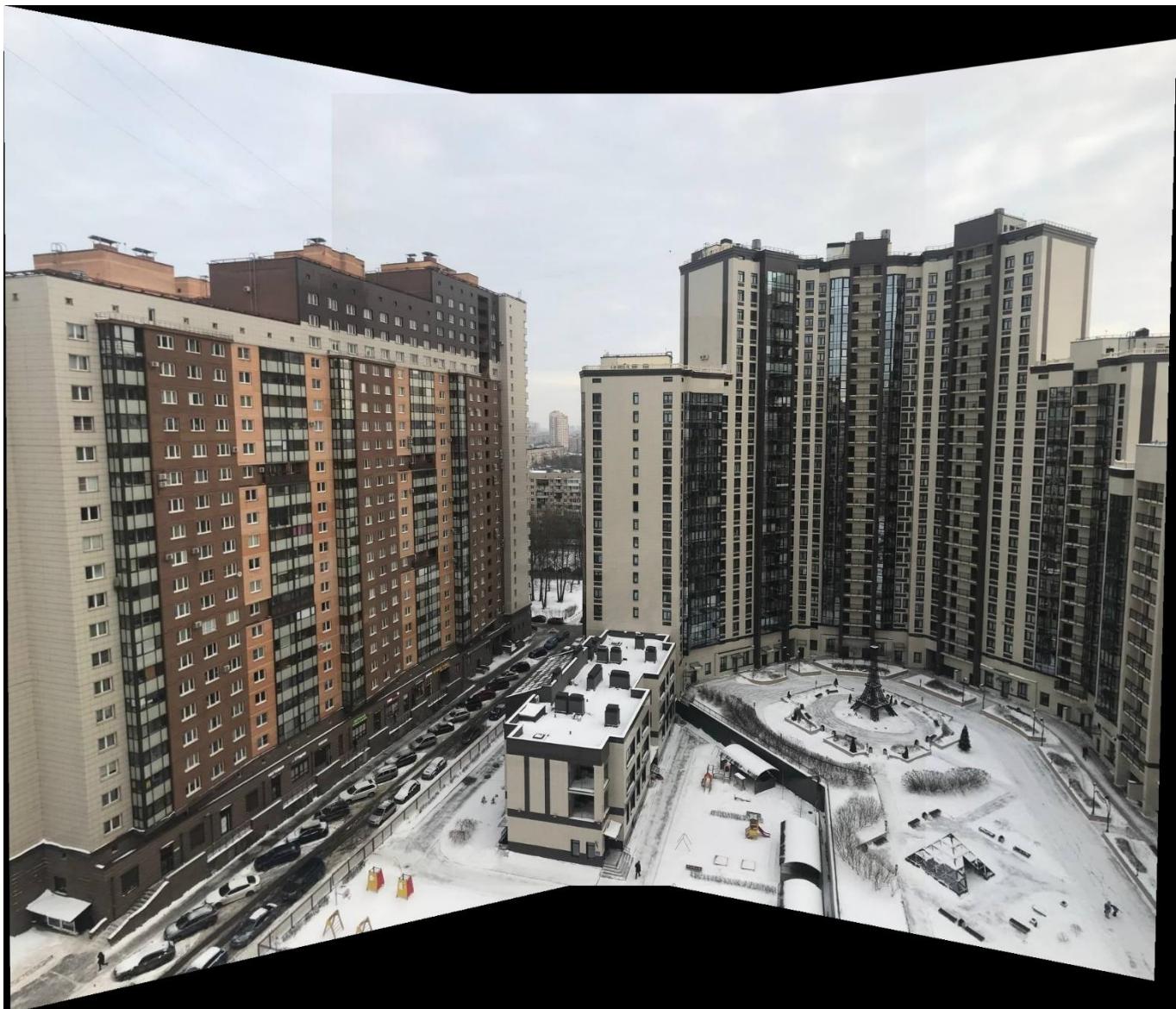


Figure 52: результат работы программы.

Результат работы программы отличный, 3 изображения склеились в одно панорамное.

Программа действительно удовлетворяет условию «автоматический», так как может без изменения кода, выполнять горизонтальное склеивание в панораму 3-х изображений, имеющих порядок.

Протестируем данную программу другим множеством изображений.



Figure 53: изображение 1.

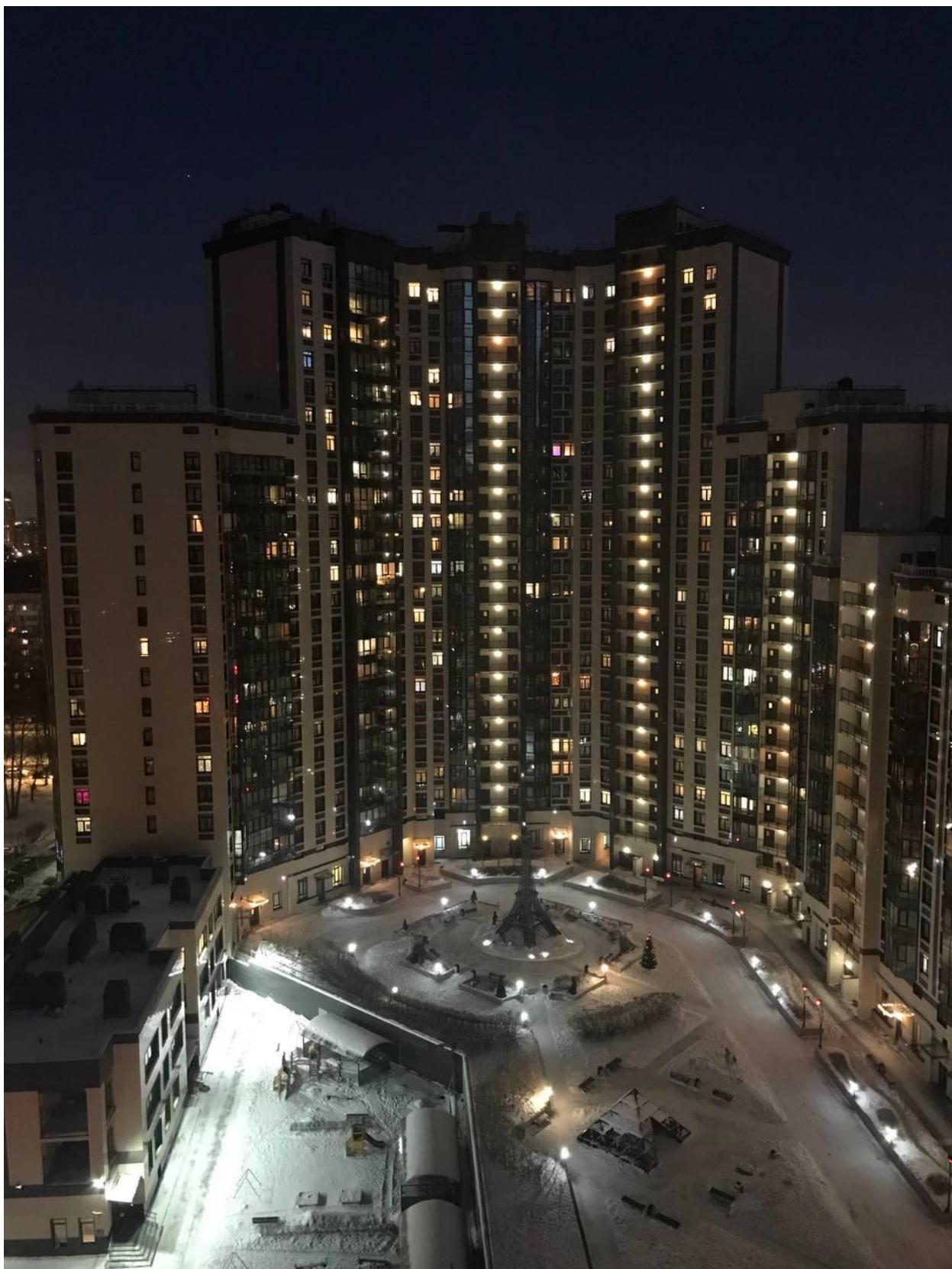


Figure 54: изображение 2.



Figure 55: изображение 3.

Предоставлю только результат работы, без промежуточных действий.

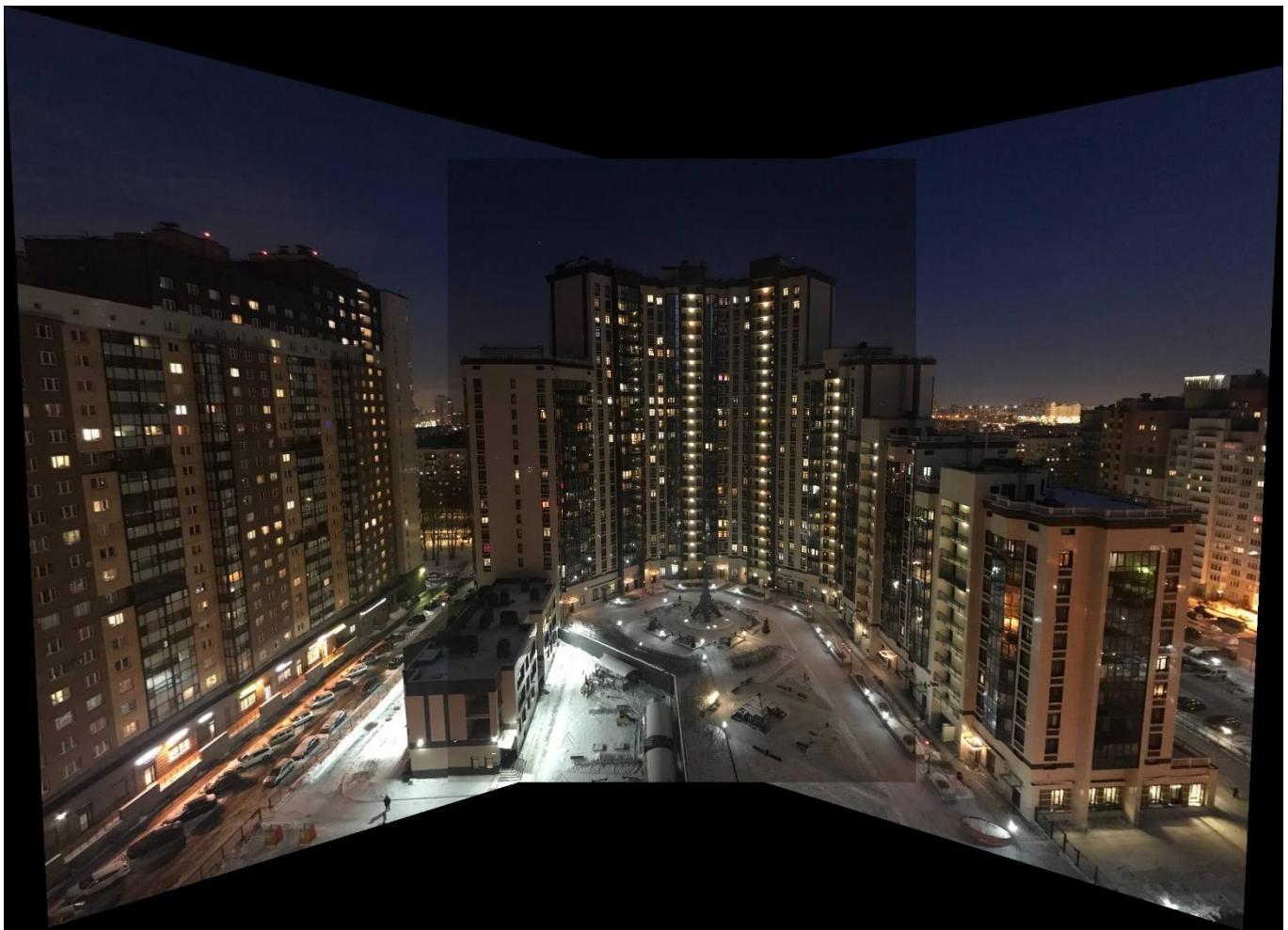


Figure 56: результат склеивания изображений.

Склейвание произошло успешно и на данных изображениях.

Выводы

В данной лабораторной работе были применены на практике детекторы *SIFT*, *ORB* и дескрипторы *SIFT*, *BRIEF*.

Для поиска характеристических точек подходят оба детектора, однако результаты дают существенно разные. Исходя из этого можно сделать вывод, что применение того или иного детектора (как и дескриптора) должно быть оправдано той задачей, которая стоит перед исследователем.

В второй части работы была проведена работа по детекции, описании и сопоставлении характеристических точек объекта и сцены, на которой присутствует объект. Пользуясь алгоритмом *RANSAC* была найдена матрица перспективного преобразования одного изображения к другому, таким образом можно найти объект, который имеет другой масштаб, ориентацию, освещение и немного сдвинут с точки обзора.

В последней части было реализовано автоматическое склеивание 3-х изображений (с порядком) в одно панорамное. Использовались методы, изученные на лекциях и в первых двух частях работы.

Результаты данной программы удовлетворительные.