Склеивание панорам

Константин Кожемяков, Федор Швецов, Владимир Лютов, Влад Шахуро

В данном задании нужно из нескольких кадров, снятых из одной точки, но под разными углами, составить одну панораму. Для этого мы будем использовать дескрипторы особых точек ORB и алгоритм оценки параметров модели RANSAC. Для более качественного склеивания мы также воспользуемся пирамидой Лапласа. Итоговую панораму будем выравнивать с помощью цилиндрической проекции.

Структура задания

В папке с заданием содержится:

1. `plots.py` В этом файле содержатся вспомогательные функции для визуализации задания.
2. `main.py` В этом файле содержится код для визуализации, разбитый по частям задания.
3. `panorama.py` Это основной файл, с которым вам придётся работать. Здесь содержатся все прототипы функций, которые вам надо реализовать, а также несколько уже написанных функций.
4. `run.py` Это вспомогательный файл для проведения тестов.
5. `tests/` Это папка с тестами.
6. `results/` Это папка для склееных панорам. Туда сохранит итоговые картинки `main.py`.
7. `imgs/` Это папка со всеми необходимыми изображениями.
8. `panorama.pdf` Это то, что вы сейчас читаете.

Критерии оценки

Максимальная оценка за задание — 10 баллов. За пункты 2,3,6,7 можно получить по 2 балла. За 4 и 5 пункты можно получить по 1 баллу.
Для пунктов 2 и 3 есть тесты. Для этих пунктов запрещается использовать готовые реализации из `skimage`. Остальные пункты проверяются вручную.

1. Выделение ключевых точек на изображении

Для начала найдем ключевые точки на каждом кадре. Если на последующих этапах качество совмещения будет недостаточным, настройте количество точек, которые находит алгоритм ORB. Вам необходимо написать функцию `find_orb(img, n_keypoints)`. Для параметра `n_keypoints` задайте подходящее значение по умолчанию.

2. Вычисление матрицы гомографии

Напишите функцию `find_homography(src_keypoints, dest_keypoints)`, которая вычисляет матрицу гомографии, по двум наборам соответствующих точек.

Для дальнейших вычислений мы будем использовать гомогенные координаты, так как они позволяют представить любое преобразование как матрицу 3×3.

Для начала нам нужно нормализовать данные наборы точек, поэтому напишите функцию `center_and_normalize_points(points)`, которая вычисляет матрицу нормирующего преобразования и отнормированные точки. Эта преобразование должно сдвигать центр точек в начало координат, а также делать их среднее расстояние до центра равным $\sqrt{2}$. Матрица для него выглядит так:

$$M = \begin{bmatrix} N & 0 & -NC_x \\ 0 & N & -NC_y \\ 0 & 0 & 1 \end{bmatrix}$$

где N - нормировочный коэффициент, а $\mathbf{C}$ - центр точек. Получившиеся матрицы для `src_keypoints`, `dest_keypoints` соответственно назовём M и M'.

Теперь вернёмся к основной функции. Координаты точек связаны соотношением

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \Leftrightarrow \mathbf{x}_2 = H\mathbf{x}_1$$

H - это как раз наша матрица гомографии.

Ингомогенные (обычные) координаты $x_2' = x_2/z_2$ и $y_2' = y_2/z_2$ получаются как

$$x_2' = \frac{H_{11}x_1 + H_{12}y_1 + H_{13}z_1}{H_{31}x_1 + H_{32}y_1 + H_{33}z_1}$$

$$y_2' = \frac{H_{21}x_1 + H_{22}y_1 + H_{23}z_1}{H_{31}x_1 + H_{32}y_1 + H_{33}z_1}$$

так как, $z_1 = 1$, верно

$$x_2'(H_{31}x_1 + H_{32}y_1 + H_{33}) = H_{11}x_1 + H_{12}y_1 + H_{13}$$

$$y_2'(H_{31}x_1 + H_{32}y_1 + H_{33}) = H_{21}x_1 + H_{22}y_1 + H_{23}$$

тогда, переставив местами слагаемые, мы можем получить два уравнения:

$$\mathbf{a}_x^T\mathbf{h} = \mathbf{0}$$

$$\mathbf{a}_x^T\mathbf{h} = \mathbf{0}$$

где

$$\mathbf{h} = (H_{11}, H_{12}, H_{13}, H_{21}, H_{22}, H_{23}, H_{31}, H_{32}, H_{33})^T$$

$$\mathbf{a}_x = (-x_1, -y_1, -1, 0, 0, 0, x_2'x_1, x_2'y_1, x_2')^T$$

$$\mathbf{a}_y = (0, 0, 0, -x_1, -y_1, -1, y_2'x_1, y_2'y_1, y_2')^T$$

Использовав все N данные нам пары точек (должно быть как минимум 4), мы получаем систему

$$A\mathbf{h} = \mathbf{0}$$

где

$$A = \begin{pmatrix} \mathbf{a}_{x1}^T \\ \mathbf{a}_{y1}^T \\ \vdots \\ \mathbf{a}_{xN}^T \\ \mathbf{a}_{yN}^T \end{pmatrix}$$

Найти решение этой однородной системы можно с помощью SVD (сингулярного) разложения матрицы $A = U\Sigma V^T$. Последняя строка из матрицы V^T и будет нашим решением $\mathbf{h}$.
Пояснение: Так как мы хотим найти ненулевое решение однородной системы, то любое решение вида $k\mathbf{h}$ нам подойдёт. Поэтому ограничим $\|\mathbf{h}\| = 1$. Тогда мы минимизируем $\|U\Sigma V^T\mathbf{h}\|$ при условии $\|\mathbf{h}\| = 1$. Так как матрицы U, V^T - унитарные, то верно, что $\|U\Sigma V^T\mathbf{h}\| = \|\Sigma V^T\mathbf{h}\|$ и $\|\mathbf{h}\| = \|V^T\mathbf{h}\|$. Сделав замену $\mathbf{y} = V^T\mathbf{h}$, мы получим задачу минимизации $\|\Sigma\mathbf{y}\|$, при условии $\|\mathbf{y}\| = 1$. Так как матрица Σ диагональная, и значения на диагонали упорядоченно убывают, решением этой системы будет вектор $\mathbf{y} = (0, 0, \ldots, 0, 1)^T$. Тогда решением исходной задачи будет $\mathbf{h} = V\mathbf{y}$, то есть последний столбец матрицы V (или последняя строка из матрицы V^T).

И в конце, используя матрицы нормировки, наша матрица гомографии H' получится как

$$H' = M'^{-1}HM$$

Проверьте реализацию с помощью юнит-тестов:
`$ ./run.py unittest homography`

3. Нахождение преобразований с помощью RANSAC

Напишите функцию `ransac_transform(src_keypoints, src_descriptors, dest_keypoints, dest_descriptors, max_trials, residual_threshold, return_matches=False)`. Для параметров `max_trials`, `residual_threshold` подберите значения и задайте их в качестве значений по умолчанию. Функция получает на вход два набора точек и соответствующих им дескрипторов. Возвратить же она должна преобразование между двумя кадрами.

Для начала вам нужно найти приблизительное соответствия между точками простым перебором, для этого можете использовать функцию `skimage.feature.match_descriptors`.

Затем вам необходимо реализовать алгоритм RANSAC для нахождения точных соответствий «инлаеров» и отбраковывания выбросов «аутлаеров».

RANSAC:

1. Повторять `max_trials` раз
   (a) Выбрать случайно 4 пары точек из всего набора.
   (b) Вычислить по ним матрицу гомографии. (Здесь вам как раз пригодится функция, написанная в прошлом пункте).
   (c) Посчитать количество точек, которые удовлетворяют полученному преобразованию. Для этого для каждой точки надо проверить, получается ли расстояние между преобразованной точкой из первого изображения и точкой из второго изображения меньше чем `residual_threshold`.
   (d) Если количество «инлаеров» для данного преобразования больше, чем у текущего лучшего преобразования, то сохранить эти хорошие точки.

2. Посчитать ещё раз матрицу гомографии, но уже используя все «инлаеры».

Проверьте решение с помощью тестов:
`$ ./run.py unittest ransac`

4. Преобразование всех кадров в плоскость центрального

Теперь когда у нас есть все преобразования между соседними кадрами, найдём преобразования всех кадров на плоскость центрального кадра.

1. Для этого напишите функцию `find_simple_center_warps(forward_transforms)`. Помните, что преобразования из `skimage` можно складывать. Также вам понадобится функция `numpy.linalg.inv`.

2. После визуализации вы можете наблюдать, что часть изображения ушла в отрицательную часть плоскости, а часть вышла за рамки центрального изображения. Если мы сейчас начнем соединять изображения, то всё за рамками центрального изображения будет обрезано. Поэтому напишите функцию `get_final_center_warps(image_collection, simple_center_warps):`, которая решает эту проблему. В ней нужно:

   (a) Получить такие преобразования, которые отправляли бы изображение в положительную четверть.

   (b) Рассчитать размер финального изображения.

   Вам могут помочь уже написанные функции: `get_corners` и `get_min_max_coords`.

5. Совмещение всех кадров на изображении

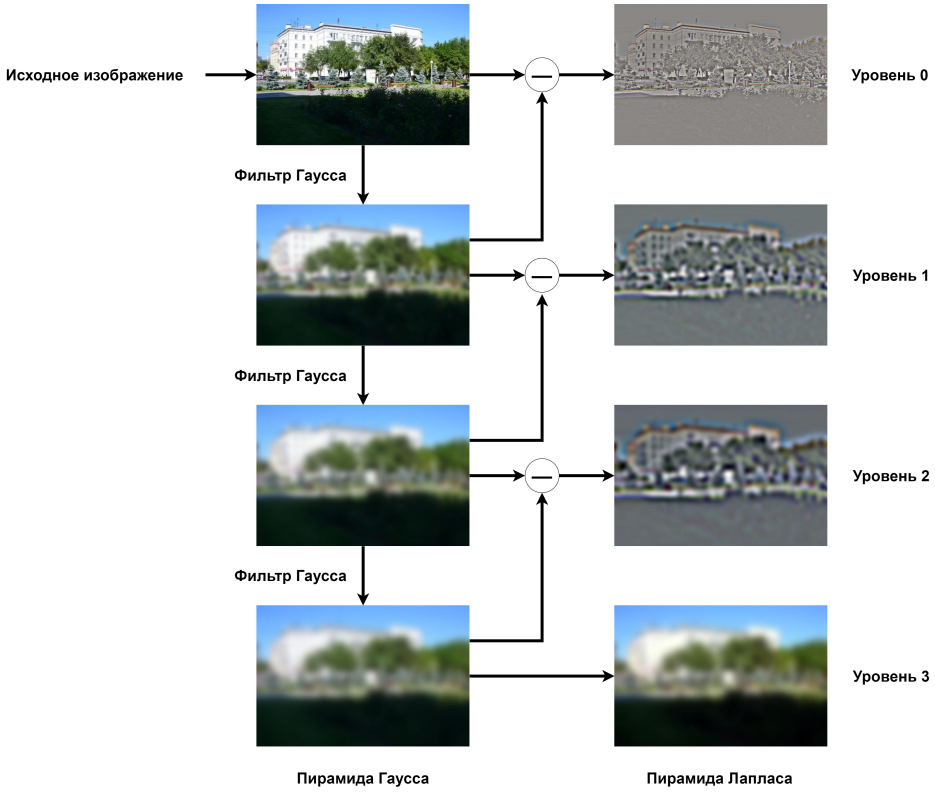
Теперь пришло время соединить все изображения в одну панораму.

1. Сначала напишите вспомогательную функцию `warp_image(image, transform, output_shape)`, которая преобразовывает изображение, а также преобразовывает его маску. Обратите внимание, что вам потребуется функция `rotate_transform_matrix(transform)`, которая поворачивает матрицу преобразования, т.к. она рассчитывается в координатах yx, а функция warp из skimage работа с изображениями в координатах xy.

2. Теперь для итоговой склейки панорамы напишите функцию `merge_pano(image_collection, final_center_warps, output_shape)`. В ней вы в цикле на вход вы получаете новый кадр и его маску, а также результат и его маску. Вам необходимо обновить результат и маску.

У вас должен получиться похожий результат.

6. Склеиваем панораму с помощью пирамиды Лапласа



1. Напишите функцию `get_gaussian_pyramid(image, n_layers, sigma)`, которая должна возвращать пирамиду Гаусса состоящую из изображений (слоёв) и полученную с помощью последовательного размытия исходного изображения фильтром Гаусса.

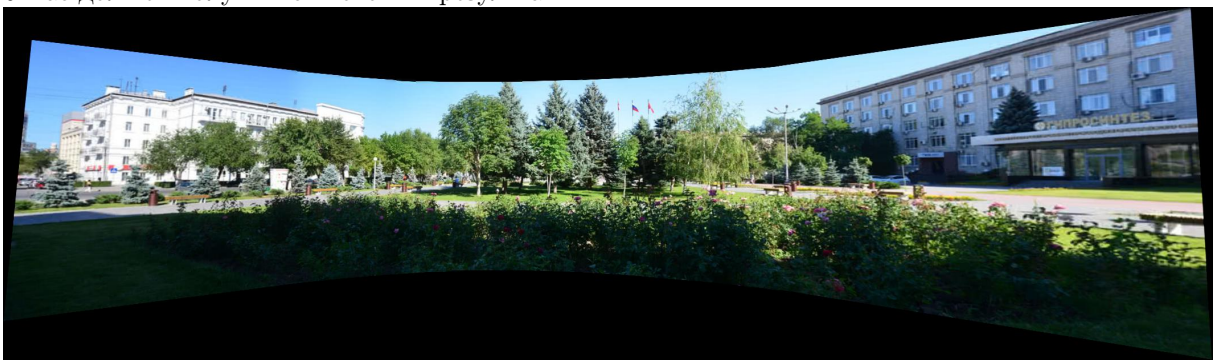
2. Затем напишите функцию `get_laplacian_pyramid(image, n_layers, sigma)` которая возвращает уже пирамиду Лапласа. Сумма всех уровней пирамиды Лапласа должна давать исходное изображение. Проверьте это в main.py.

3. Ну и напоследок напишите функцию `gaussian_merge_pano(image_collection, final_center_warps, output_shape, n_layers, image_sigma, merge_sigma)`. Эта функция объединяет все полученные нами вещи и возвращает уже качественно склеенную панораму.

   - Для начала разберемся с масками. В предыдущем пункте у нас получались дыры между кадрами, потому что на каждом следующем шаге мы накладывали новое изображение *целиком* на результат предыдущего шага. Из-за этого все границы текущего изображения получали маску, равную 1, хотя могли быть искажены. Поэтому теперь лучше *заранее* разделить пространство итогового изображения между всеми кадрами. Например, половину каждого пересечения двух соседних кадров отдать левому кадру, а вторую половину отдать правому кадру. Тогда сами *границы* обоих изображений получат маску 0 и не будут вызывать искажения.
   - Дыр между кадрами больше нет, но швы все еще заметны. Хотелось бы размыть панораму на стыках. Для этого будем использовать пирамиды Гаусса и Лапласа.
     - Из масок сделаем пирамиды Гаусса с `merge_sigma`. Получится $N =$ `len(image_collection)` пирамид высоты $L =$ `n_layers`. Маски теперь не бинарные, а вещественные со значениями от 0 до 1. Так как из-за размытия маски соседних кадров снова пересекаются,

то желательно нормализовать каждый уровень пирамиды: сумма N масок на каждом уровне должна давать только 0 или 1, иначе будут заметные скачки яркости на швах.
  - Преобразуем все кадры с помощью нашей функции `warp_image`. Из преобразованных кадров сделаем пирамиды Лапласа с `image_sigma`. В итоге получим N пирамид Лапласа высоты L.
  - Перемножим все кадры на всех уровнях из пирамиды Лапласа с соответствующими масками из пирамиды Гаусса. Сложив полученные $N \cdot L$ результатов получим качественно склеенную итоговую панораму.
- Поэкспериментируйте с различными значениями уровней пирамиды и sigma. Начните с пары уровней пирамиды и небольших значений sigma, чтобы самостоятельно заметить описанные выше проблемы. Увеличивайте эти параметры и добейтесь того, чтобы переходы при склейке были незаметны. Сильно размывать изображение (`image_sigma`) не стоит, а размытие масок (`merge_sigma`) можно сделать сильным. Достаточно 3-4 слоев пирамид.

У вас должен получиться похожий результат.



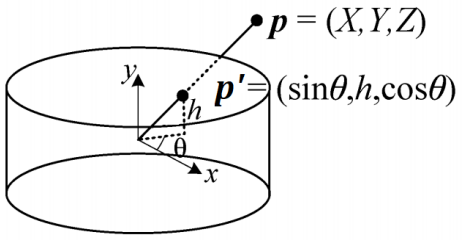
7. Выравнивание изображений с помощью цилиндрической проекции

Итоговое изображение можно выровнять с помощью цилиндрической проекции, чтобы сделать изображение панорамы более прямоугольным.

Реализуйте функцию `warp_cylindrical(img)`, преобразующую изображение в цилиндрическую проекцию.

Для начала рассмотрим теоретическое обоснование. Пусть координаты пикселей на **исходном изображении** — (x, y). А на **новом изображении** — (θ, h), где θ — угол в горизонтальной плоскости, h — вертикальная высота в цилиндре (см. изображение).

Будем работать с гомогенными координатами. Рассмотрим точку с координатами $\boldsymbol{p} = (X, Y, Z)$. Точка пересечения луча $\boldsymbol{p}$ с боковой поверхностью цилиндра с радиусом 1 имеет координаты $\boldsymbol{p'} = (\sin\theta, h, \cos\theta)$.



Выразим θ, h:

$$\theta = \arctan\frac{X}{Z},\ h = \frac{Y}{\sqrt{X^2 + Z^2}}$$

Так как $Z = 1$:

$$\theta = \arctan X,\ h = \frac{Y}{\sqrt{X^2 + 1}}$$

Отсюда:

$$X = \tan\theta$$

$$Y = h\sqrt{X^2+1} = h \cdot \sqrt{\tan^2\theta + 1} = \frac{h}{\cos\theta}$$

Таким образом, мы можем поставить в соответствие любому пикселю с координатами $(x', y') = (\theta, h)$ на новом изображении некоторый пиксель с координатами $(x, y) = (\tan\theta, \frac{h}{\cos\theta})$ на исходном изображении. Для преобразования изображения воспользуйтесь функцией `skimage.transform.warp`, но уже с кастомным преобразованием (см. второй аргумент `inverse_map`).

Чтобы формулы имели смысл, необходимо $\theta \in (-\frac{\pi}{2}, \frac{\pi}{2})$. Поэтому необходимо преобразовать координаты изображений: перенести точку $(0, 0)$ в центр изображения и поделить координаты на $S = \frac{W}{\pi}$. *Замечание: параметр S отвечает за масштаб цилиндра, его можно выбирать любым в интервале $(\frac{W}{\pi}, +\infty)$. Меньшие значения S будут сильнее деформировать исходные изображения.* Будем использовать матрицу

$$K = \begin{bmatrix} S & 0 & W/2 \\ 0 & S & H/2 \\ 0 & 0 & 1 \end{bmatrix}$$

Итого, функция `warp_cylindrical(img)` должна делать следующее:

1. Создать матрицу $C \in \mathbb{R}^{3\times H\cdot W}$ гомогенных координат всех пикселей выходного изображения. C_x и C_y это x'-координаты и y'-координаты соответственно, I — строчка из единиц.

$$C = \begin{bmatrix} x'_0 & x'_1 & \dots \\ y'_0 & y'_1 & \dots \\ 1 & 1 & \dots \end{bmatrix} = \begin{bmatrix} C_x \\ C_y \\ I \end{bmatrix}$$

2. Центрировать и нормализовать координаты: $\hat{C} = K^{-1} \cdot C$

3. Перевести координаты в цилиндрическую проекцию: $\hat{B}_x = \tan\hat{C}_x$, $\hat{B}_y = \frac{\hat{C}_y}{\cos\hat{C}_x}$ (все операции поэлементные)

4. Сделать обратную нормализацию, вычислить B_x, B_y:

$$\begin{bmatrix} B_x \\ B_y \\ I \end{bmatrix} = K \cdot \begin{bmatrix} \hat{B}_x \\ \hat{B}_y \\ I \end{bmatrix}$$

5. Преобразовать изображение с помощью `skimage.transform.warp`, используя B_x, B_y

У получающегося изображения можно отрезать прямоугольные черные полосы по краям. Поэкспериментируйте с разными параметрами S для получения более ровной картинки. У вас должен получиться похожий результат.