Совмещение каналов изображения

Стоцкий Андрей, Константин Кожемяков,
Влад Шахуро, Александр Сергеев

Graphics & Media Lab
Vision Group

Обзор задания

Первым цветным фотографом России является Сергей Михайлович Прокудин-Горский, сделавший единственный цветной портет Льва Толстого. Каждый его снимок представляет из себя три изображения в градациях серого, соответствующие синему, зеленому и красному цветовым каналам. Сейчас коллекция его снимков находится в американской библиотеке конгресса, сканкопии фотопластинок доступны в интернете. В данном задании мы предлагаем вам создать программу, которая будет совмещать изображения, полученные с фотопластинок Прокудина-Горского.

Критерии оценки

Максимальная оценка за задание — 5 баллов. Разбиение этих баллов между разными юнит-тестами, можно посмотреть в файле `run.py`, в списке `grade_mapping`. Для выполнения задания нужно скачать из проверяющей системы:

1. Шаблон решения — архив с файлом `align.py`.
2. Архив с тестами. Разархивируйте его в папку `tests`.
3. Скрипт для тестирования `run.py`. Добавьте ему права на выполнение. Для запуска тестов вам понадобится библиотека `pytest`.
4. Дополнительные файлы для решения. В данном задании это архив с файлами `common.py` и `pipeline.py`, в которых содержатся вспомогательные функции для тестов и уже написанные за вас функции, с помощью которых можно запустить ваш пайплайн на одном из тестовых изображений и визуализировать ключевые точки на изображениях.

Файлы должны располагаться следующим образом:

```
├── align.py
├── common.py
├── pipeline.py
├── run.py
└── tests/
```

Описание задания

1 Извлечение цветных пластин из исходного изображения

Как обсуждалось на лекции, фотопластинки Прокудина-Горского состоят из трех вертикально расположенных изображений в серых тонах, полученных в результате съемки сцены с синим, зеленым и красным фильтрами (именно в таком порядке).

Вам необходимо реализовать функцию `extract_channel_plates`, принимающую на вход одноканальное изображение, полученное сканированием таких фотопластинок. Функция должна возвращать кортеж из трех извлеченных каналов – красного, зеленого и синего (именно в таком порядке), а также кортеж из трех координат, задающих положение левого верхнего угла каждого из извлеченных фрагментов в координатной системе исходного изображения.

Для этого достаточно разделить изображение на три равные части по высоте. В случае, если высота не делится на три нацело, последние несколько строк удаляется.

Дополнительно, если передана опция `crop=True`, то необходимо удалить рамки фотопластинок из полученных каналов. Для этого, предлагается обрезать каждый из извлеченных каналов на 10% с каждой стороны. Не забудьте также обновить возвращаемые координаты.

Режим `crop=True` будет использоваться перед запуском алгоритмов для поиска относительных сдвигов между каналами. Без данного кропа, границы рамок фотопластинок будут мешать нахождению истинного сдвига.

Проверьте реализацию с помощью юнит-тестов:

```
$ ./run.py unittest extract_channel_plates
```

2 Поиск сдвига с помощью пирамиды изображений

Для того, чтобы совместить два извлеченных канала, будем сдвигать один канал относительно другого по горизонтали и по вертикали в некоторых пределах, например, от -15 до 15 пикселей. Далее, для перекрывающихся областей изображений посчитаем метрику.

Оптимальным будет тот сдвиг, при котором метрика принимает наибольшее или наименьшее значение (в зависимости от метрики). Предлагается реализовать две метрики и выбрать ту, которая позволяет получить более качественный результат при совмещении:

1. Среднеквадратичное отклонение для изображений I_1 и I_2:

$$MSE(I_1, I_2) = \frac{1}{width \cdot height} \sum_{x,y} (I_1(x,y) - I_2(x,y))^2,$$

   где $width, height$ – ширина и высота перекрывающихся областей изображений.
   Для нахождения оптимального сдвига нужно взять минимум по всем сдвигам.

2. Нормализованная кросс-корреляция для изображений I_1 и I_2:

$$I_1 \star I_2 = \frac{\sum_{x,y} I_1(x,y) I_2(x,y)}{\sqrt{\sum_{x,y} I_1^2(x,y) \cdot \sum_{x,y} I_2^2(x,y)}}.$$

   Для нахождения оптимального сдвига нужно взять максимум по всем сдвигам.

Совмещение больших изображений при базовом подходе будет проходить очень медленно. Для ускорения совмещения необходимо реализовать поиск сдвига по пирамиде изображений.

В пирамиде изображений исходное изображение последовательно уменьшается в 2 раза до некоторого размера (например, чтобы обе стороны были не больше 500 пикселей). Поиск оптимального сдвига начинается с самого маленького изображения, а затем на пути к исходному изображению уточняется на уменьшенных копиях изображения.

Таким образом, оригинальное большое изображение будет совмещаться не в диапазоне $-15 \dots 15$ пикселей, а в меньшем диапазоне, уже примерно определенном с использованием уменьшенных изображений.

Проверьте реализацию с помощью юнит-тестов:

```
$ ./run.py unittest find_relative_shift_pyramid
```

3 Сопоставление пластин

Реализованная в прошлом разделе функция позволяет найти относительные сдвиги между парой каналов. В этом разделе вам предлагается реализовать функцию `find_absolute_shifts`, которая находит абсолютные сдвиги красного и синего каналов по отношению к зеленому. Данная функция принимает на вход кортеж из трех каналов и координаты откуда из исходного изображения были извлечены эти каналы.

Далее, с помощью переданной функции `find_relative_shift_fn`, необходимо найти относительные сдвиги между зеленым и красным каналами, а также между зеленым и синим каналами. После этого, необходимо перевести данные сдвиги из относительных в абсолютные (то есть возвращаемые сдвиги должны задавать смещение между одинаковыми точками на разных пластинах на исходном изображении).

Проверьте реализацию с помощью юнит-тестов:

```
$ ./run.py unittest find_absolute_shifts
```

На данном этапе, вы также можете посмотреть на визуализацию промежуточных результатов.

В тестовых данных, на каждой картинке размечены три точки (по одной на каждом канале), соответствующие положению одного и того же объекта на сцене. Используя реализованные вами функции, функция `align_image` из модуля `pipeline.py` выполняет запуск полного пайплайна, а функция `visualize_point` позволяет визуализировать истинное и предсказанное положение точек на исходном изображении.

Если запустить файл с вашим решением как скрипт (`python3 align.py`), то в текущей директории должны появиться изображения `gt_visualized.png` и `pyramid_visualized.png` содержащие выше описанные визуализации. После выполнения следующих пунктов задания, вы также сможете посмотреть на сами совмещенные изображения, полученные методами `pyramid` и `fourier`.

4 Создание цветного изображения

Теперь, когда у нас реализованы функции для извлечения пластин из исходного изображения и для сопоставления их координат, осталось лишь реализовать логику для сборки цветного изображения.

Функция `create_aligned_image` должна принимать кортеж извлеченных из исходного изображения каналов, кортеж координат этих каналов и абсолютные сдвиги необходимые для совмещения этих каналов.

На выходе, данная функция должна возвращать совмещенное цветное изображение. Обратите внимание, что из-за относительных сдвигов, извлеченные каналы могут не идеально накладываться друг на друга. Возвращаемое из данной функции изображение должно содержать те и только те пиксели, которые присутствуют во всех трех каналах.

Проверьте реализацию с помощью юнит-тестов:

```
$ ./run.py unittest create_aligned_image
```

5 Проверка работы пайплайна на реальных изображениях

Во всех предыдущих тестах работа ваших функций проверялась на простых синтетических данных. Теперь, когда вы реализовали все необходимые шаги в пайплайне для совмещения изображений, пришло время проверить работоспособность вашей реализации на реальных изображениях Прокудина-Горского.

В следующих тестах проверяется работа вашей реализации на сканах 10 настоящих изображений, снятых Прокудиным-Горским. Тесты разделены на две группы – в зависимости от разрешения этих изображений (`small` и `large`).

Для предсказанных координат точек и координат разметки вычисляется метрика l_1, которая затем сравнивается с порогом. Если метрика не превосходит порог, то изображение считается качественно совмещенным. Для маленьких изображений порог равен 5, для больших – 10.

Проверьте реализацию с помощью юнит-тестов:

```
$ ./run.py unittest align_image_pyramid_img_small
```

```
$ ./run.py unittest align_image_pyramid_img_large
```

6 Поиск сдвига с помощью преобразования Фурье

Когда ищутся достаточно большие сдвиги (например, при сшивании изображений или поиске части изображения на целом), пирамидальный подход может работать не очень хорошо: для бóльших сдвигов нужно сильнее уменьшать исходные изображения, что приводит к потере деталей. В этом случае хорошо работает подход, основанный на преобразовании Фурье.

Рассмотрим нормализованную кросс-корреляцию для двух изображений I_1 и I_2. Для нахождения оптимального сдвига нужно взять максимум по всем сдвигам:

$$(u^*, v^*) = \underset{u,v}{\operatorname{argmax}} \frac{\sum_{x,y} I_1(x,y) I_2(x+u, y+v)}{\sqrt{\sum_{x,y} I_1^2(x,y) \cdot \sum_{x,y} I_2^2(x,y)}}.$$

Для сдвига изображения будем использовать циклический сдвиг. Тогда, выражение в знаменателе становится константой и на значение argmax не влияет:

$$(u^*, v^*) = \underset{u,v}{\operatorname{argmax}} \sum_{x,y} I_1(x,y) I_2(x+u, y+v).$$

Заметим, что справа находится операция корреляции. Преобразование Фурье позволяет заменить дорогую операцию корреляции на более дешевую операцию поэлементного произведения. Обозначим

$$C(u,v) = \sum_{x,y} I_1(x,y) I_2(x+u, y+v) = (I_1 * I_2)(u,v).$$

Матрица C содержит значения кросс-корреляции двух изображений *для всех возможных сдвигов*. Координаты максимального значения матрицы C и есть искомый сдвиг. Вычислить эффективным образом матрицу C можно с помощью преобразования Фурье:

$$\mathcal{F}\{C(u,v)\} = \mathcal{F}\{(I_1 * I_2)(u,v)\} = \left(\overline{\mathcal{F}\{I_1\}} \cdot \mathcal{F}\{I_2\}\right)(u,v),$$

$$C(u,v) = \mathcal{F}^{-1}\left(\overline{\mathcal{F}\{I_1\}} \cdot \mathcal{F}\{I_2\}\right)(u,v).$$

Здесь $\mathcal{F}$ и $\mathcal{F}^{-1}$ – прямое и обратное преобразование Фурье, $\overline{x}$ – комплексное сопряжение, а $\cdot$ – поэлементное произведение. Таким образом, для нахождения оптимального сдвига необходимо вычислить

$$\operatorname{argmax} \mathcal{F}^{-1}\left(\overline{\mathcal{F}\{I_1\}} \cdot \mathcal{F}\{I_2\}\right).$$

Проверьте реализацию с помощью юнит-тестов – на синтетических данных:

```
$ ./run.py unittest find_relative_shift_fourier
```

и на реальных изображениях:

```
$ ./run.py unittest align_image_fourier_img_small
```

```
$ ./run.py unittest align_image_fourier_img_large
```

Полезные ресурсы

Выставка о Прокудине-Горском на сайте библиотеки конгресса.