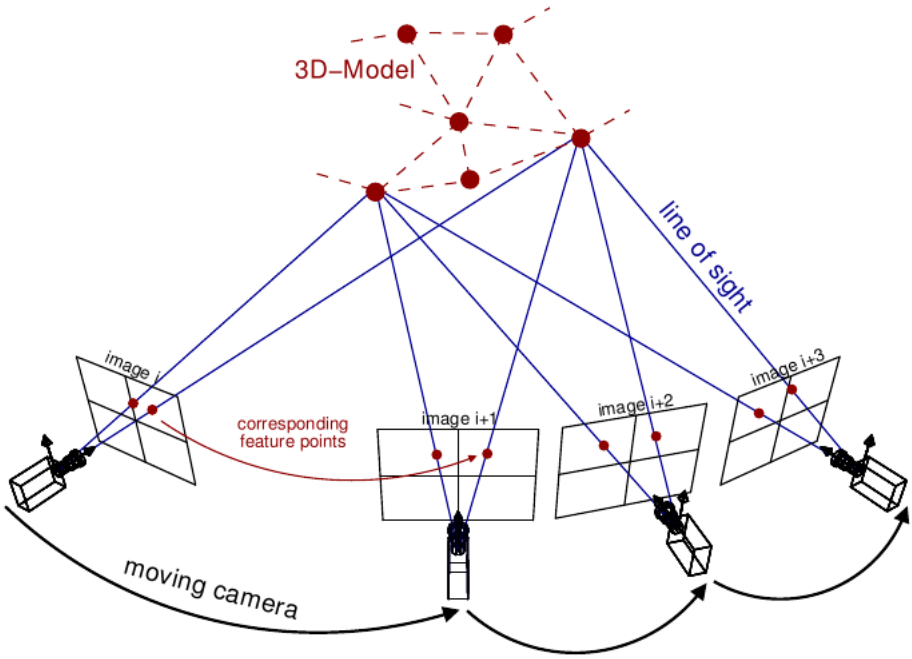
Восстановление положения камер и трехмерной структуры сцены

Александр Велижев — исследовательский центр IBM
Тимур Ибадов, Иван Малин — GeoCV

Graphics & Media Lab
Vision Group

GeoCV

Введение



Данное задание посвящено теме трехмерной реконструкции сцены на основе фотографий с движущейся камеры. Эту область обычно называют «восстановление структуры сцены из движения» (Structure-from-Motion, SfM) или «одновременная локализация и картографирование сцены» (Simultaneous Localization and Mapping, SLAM). Иногда еще используют термины визуальная геометрия (Visual Geometry) или визуальная одометрия (Visual Odometry). Для краткости мы будем использовать термин SLAM в дальнейшем (отметим, что эти термины не совсем идентичны, хотя и близки по смыслу).

Методы SLAM работают с разными сенсорами или их комбинациями. Минимальный вариант включает одну движущуюся камеру (monocular SLAM). Однако возможны и варианты для стереокамер (stereo cameras) или сенсоров глубины (depth sensor). Использование разных сенсоров вместе частично устраняет их отдельные недостатки и позволяет сделать решение более надежным. Часто в SLAM добавляют данные систем спутниковой навигации (GPS) или датчиков скорости и ускорения (Inertial Measurement Units, IMU). В данном задании вы будете работать с классическим вариантом SLAM для одной камеры.

В настоящее время данная область активно развивается и уже применяется на практике. Например, SLAM лежит в основе систем навигации автопилота автомобилей. Другой пример — методы

дополненной реальности используют методы SLAM для правильного рендеринга виртуальных объектов в зависимости от положения наблюдателя. Последние модели роботов-пылесосов имеют встроенную камеру, которая позволяет роботу точно определить свое положение в квартире. В области дронов SLAM позволяет избегать столкновений с препятствиями и точно двигаться по заданной траектории. Надеюсь, что мы убедили вас в том, что тема SLAM актуальна и современна — идем дальше!

Зачем мне нужно делать это задание?

SLAM является важной частью компьютерного зрения, и нельзя стать профессионалом в этой области, не понимая принципов его работы. Текущее задание включает использование самых базовых методов, которые лежат в основе SLAM.

Описание задачи

Реализация полноценной версии SLAM достаточно трудоемка, поэтому мы сделали для вас несколько искусственное задание, которое, однако, позволит понять, как же работает SLAM. Кроме того, вы можете использовать готовые реализации базовых алгоритмов, чтобы собрать из них ограниченную версию SLAM, которая отлично подходит для учебных целей. На входе задачи имеется набор из N изображений, полученных с одной камеры. Примерно для половины изображений даны координаты центров проекции камер и матрицы поворота. **На выходе вам нужно определить положения оставшейся части камер**. Кроме того, вам будут даны параметры внутренней калибровки камер.

Давайте определимся с терминами. Камеры, для которых положение известно, мы будем называть опорными, оставшиеся камеры — неизвестными. Чтобы усложнить задачу, мы добавили шум в положения опорных камер, поэтому оно известно только с относительной точностью.

Итак, для решения задачи требуется реализация следующих шагов:

1. На каждом опорном и неизвестном изображении найти положение характерных точек (keypoints) и вычислить для них дескрипторы ORB. Советуем вам сохранить дескрипторы в файлы с помощью pickle, чтобы каждый раз при тестировании не вычислять (для избежания ошибок при сдаче задания советуем вычислять дескрипторы, только если ваш файл с дескрипторами не существует). Количество характерных точек на одном изрбражении должно быть в пределах нескольких сотен.

2. Последовательно для каждой пары опорных камер:

   (a) Выполнить сопоставление характерных точек в пространстве дескрипторов, включая фильтрацию по отношению расстояний между первым и вторым ближайшими дескрипторами.

   (b) Отфильтровать неверные соответствия с помощью фундаментальной матрицы и алгоритма RANSAC.

   (c) Запомнить все инлаеры для данной пары изображений.

3. Построить треки точек для всех инлаеров на всех изображениях. Например, у нас есть 3 изображения. На паре изображений 1 и 2 нашлось соответствие между точками с индексами m и n соответственно, кроме того для точки n на изображении 2 еще нашлась соответствующая точка k на изображении 3. В этом случае треком будет считаться последовательность точек m, n и k на соответствующих изображениях. По сути трек — это положение одной и той же трехмерной точки на разных изображениях.

4. Триангулировать координаты 3д точки для каждого трека. Здесь вам понадобятся положения опорных кадров, которые даны в задании. После этого шага у вас будут вычислены положения 3д точек сцены.

5. Каждую трехмерную точку проецировать на все изображения, на которых она видна, и вычислить ошибку репроекции (расстояние между положением дескриптора и репроекцией 3д точки). Если ошибка больше 10 пикселей — исключаем 3д точку (и весь трек) из дальнейшего рассмотрения.

6. Для каждого неизвестного изображения:

   (a) Провести матчинг дескрипторов относительно опорных кадров (аналогично шагу 2, но только с инлаерами на опорных кадрах). В результаты вы получите соответствия точек на опорных кадрах и текущем неизвестном. Для точек на опорных изображениях мы уже знает трехмерные координаты точек — этого достаточно, чтобы найти искомое положение неизвестных кадров.

7. Сохранить все найденные положения неизвестных камер. Это минимальное решение для всего задания.

Задание нужно реализовать на python с использованием библиотеки OpenCV и OpenCV Contrib.

Формат данных

Входные данные состоят из набора цветных изображений, полученных с одной камеры, параметров внутренней калибровки этой камеры и поз для некоторых изображений. Список цветных кадров хранится в файле `rgb.txt` в формате:
`frame_id relative_path_to_image`

Положения опорных камер находятся в файле `known_poses.txt` в формате:
`frame_id tx ty tz qx qy qz qw`

Калибровка в файле `intrinsics.txt` в следующем формате:
`focal_x focal_y center_x center_y`

Кадры с известными положениями находятся в папке `rgb_with_poses` и представляют из себя каждый второй кадр исходной видеопоследовательности. Остальные кадры, положение которых вам предстоит найти, перемешаны и находятся в папке `rgb`.

Критерии оценки

Вам требуется реализовать функцию `estimate_trajectory`, которая принимает на вход путь к входным данным и папку для сохранения результатов, в которой нужно создать файл `all_poses.txt` с положениями всех кадров в том же формате, что и `known_poses.txt` (и с теми же id кадров).

Максимальная оценка за задание — 10 баллов. Задание проверяется на 3 открытых и на 3 скрытых наборах данных. В проверяющую систему нужно сдать файл `estimate_trajectory.py`. Для оценки результатов сравниваются положения неизвестных камер с их правильными значениями (ground truth):

1. Вычисляется количество камер, положение которых определено с достаточной точностью (отклонение от ground truth по положению не более 20см и углы поворота отличаются не более, чем на 15 градусов).

2. Количество правильных положений должно быть не менее 70% от общего количества камер.

3. Для каждого тестового набора вычисляется стандартное отклонение от ground truth.

4. Итоговая оценка зависит от наибольшего из отклонений по всем тестовым наборам и от количества правильных положений камер. Подробнее алгоритм вычисления оценки можно посмотреть в функции `grade` в скрипте `run.py`.

Описание файлов для решения

Файлы и разархивированные zip-архивы из проверяющей системы разместите следующим образом:

```
├── estimate_trajectory.py
├── run.py
├── generate_point_cloud.py
├── tests/
│   ├── 00_test_slam_gt/
│   ├── 00_test_slam_input/
│   └── ...
└── common/
    ├── absolute_translational_error.py
    └── ...
```

Обратите внимание на следующие файлы:

- `estimate_trajectory.py`

  В этом файле вам нужно реализовать свой алгоритм. Обратите внимание, что в проверяющей системе папка с кодом решения монтируется в режиме для чтения (read-only).

- `run.py`

  Скрипт для тестирования решения. Запустить можно командой `./run.py tests` в консоли.

- `generate_point_cloud.py`

  Генерация облака точек по указанной траектории. Принимает на вход путь к набору данных, путь к папке с соответствующими данными для тестирования (`00_test_slam_gt`, например), путь к файлу с позами кадров и путь к итоговому облаку точек в формате PLY.

- `common/dataset.py`, `common/intrinsics.py`, `common/trajectory.py`

  Содержат функции для загрузки/сохранения разных форматов.

Дополнительная информация

Хорошая обзорная статья про текущее состояние и будущее SLAM — Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age. При подготовке задания использовались данные из набора TU Munich RGB-D SLAM Dataset.

Советы

Опыт проверки заданий показал заметную корреляцию между качеством кода решения и качеством результатов. Ниже приведен небольшой список советов.

1. Организуйте код в виде небольших коротких функций, каждая из которых решает только одну задачу. Избегайте глубокой вложенности циклов и условий. Это упростит читаемость кода, и, как следствие, сократит число ошибок.

2. Если в реализации есть части, которые требуют значительных вычислений, то реализуйте сохранение промежуточных результатов в файлы с возможностью их загрузки на лету. Например, можно вычислить все ключевые точки и дескрипторы один раз, и не пересчитывать их при каждом перезапуске программы. Однако, важно быть аккуратными и не забывать пересчетывать сохраненные файлы при изменении параметров.

3. Потратьте время на визуализацию промежуточных результатов, это поможет проверить, что все шаги работают правильно.

О задании

Задание было подготовлено при поддержке компании GeoCV, одного из лидеров в области мобильного 3D-сканирования, и Александра Велижева, научного работника исследовательского центра IBM в Цюрихе. Будем рады вашим отзывам — пишите на velizhev@gmail.com.

Приложение 1. Скрипт, иллюстрирующий триангуляцию и репроекцию 3D точки

```
import numpy as np
import cv2


def main():
    # an image point was measured on these two images
    # (the bottom left corner of the monitor)
    image_0_name = "tests/00_test_slam_input/rgb_with_poses/000000.png"
    image_6_name = "tests/00_test_slam_input/rgb_with_poses/000006.png"

    # image coordinates (measured manually)
    image_0_2d_point = np.array([305.0, 292.0], ndmin=2)
    image_6_2d_point = np.array([363.0, 195.0], ndmin=2)

    # image translation and quaternion from the file:
    # "tests/00_test_slam_gt/ground_truth.txt"
    image_0_translation = np.array([1.3390429617085644,
                                    0.6235776860273197,
                                    1.6559810006210114], ndmin=2)
    image_0_quaternion = np.array([0.6574280826403377,
                                   0.612626168885718,
                                   -0.29491259746065657,
                                   -0.32481387472099443])

    # image translation and quaternion for the second image
    image_6_translation = np.array([1.121783231448374,
                                    0.6276849491056922,
                                    1.3913821856210593], ndmin=2)
    image_6_quaternion = np.array([0.6691820728523993,
                                   0.6388828845567809,
                                   -0.2857923437256659,
                                   -0.24969331080580404]

    # compute rotation matrix from the quaternion
    # here we invert the rotation matrix because we need an inverse transfromation
    image_0_rotation = np.linalg.inv(quaternion_to_rotation_matrix(image_0_quaternion))
    # compute Rodrigues vector from rotation matrix (is needed for OpenCV)
    image_0_rodrigues, _ = cv2.Rodrigues(image_0_rotation)
    # update translation as well for the inverse transformation
    image_0_translation = -1.0 * np.matmul(image_0_rotation, image_0_translation.T)


    print("\nCamera 0 position:\n", image_0_translation)
    print("\nCamera 0 rotation matrix\n", image_0_rotation)
    print("\nCamera 0 rodrigues vector\n", image_0_rodrigues)

    # the same for the second image
    image_6_rotation = np.linalg.inv(quaternion_to_rotation_matrix(image_6_quaternion))
```

```
    image_6_rodrigues, _ = cv2.Rodrigues(image_6_rotation)
    image_6_translation = -1.0 * np.matmul(image_6_rotation, image_6_translation.T)
    print("\nCamera 6 position\n", image_6_translation)
    print("\nCamera 6 rotation matrix\n", image_6_rotation)
    print("\nCamera 6 rodrigues vector\n", image_6_rodrigues)

    # define the intrinsic matrix
    K = np.array([[525.0, 0.0, 319.5], [0.0, 525.0, 239.5], [0.0, 0.0, 1.0]])
    print("\nCamera intrinsic matrix\n", K)

    # compute projection matrices
    image_0_projection_matrix = np.matmul(K, \
                        np.concatenate((image_0_rotation, image_0_translation), axis=1))
    image_6_projection_matrix = np.matmul(K, \
                        np.concatenate((image_6_rotation, image_6_translation), axis=1))
    print("\nCamera 0 projection matrix\n", image_0_projection_matrix)
    print("\nCamera 6 projection matrix\n", image_6_projection_matrix)

    # triangulate 3D point
    point_3d = cv2.triangulatePoints(image_0_projection_matrix, \
                        image_6_projection_matrix, \
                        image_0_2d_point.T, \
                        image_6_2d_point.T)
    point_3d = point_3d / point_3d[3]
    print("\n3D point (homogeneous coordinates)\n", point_3d)

    # reproject 3D point back to the images
    image_0_2d_point_reprojection, _ = cv2.projectPoints(point_3d[0:3].T, \
                                    image_0_rodrigues, image_0_translation, K, None)
    reprojection_error_0 = np.linalg.norm(image_0_2d_point - \
                                        image_0_2d_point_reprojection)
    print("\nReprojection 2D point:\n", image_0_2d_point_reprojection)
    print("\nReprojection error for point 0: %.2f pixels\n" % reprojection_error_0)

    image_6_2d_point_reprojection, _ = \
                cv2.projectPoints(point_3d[0:3].T, \
                                    image_6_rodrigues, image_6_translation, K, None)
    reprojection_error_6 = np.linalg.norm(image_6_2d_point - \
                                        image_6_2d_point_reprojection)
    print("\nReprojection 2D point: \n", image_6_2d_point_reprojection)
    print("\nReprojection error for point 6: %.2f pixels\n" % reprojection_error_6)


def quaternion_to_rotation_matrix(quaternion):
        """
        Generate rotation matrix 3x3  from the unit quaternion.

        Input:
        qQuaternion -- tuple consisting of (qx,qy,qz,qw) where
             (qx,qy,qz,qw) is the unit quaternion.
```

```
        Output:
        matrix -- 3x3 rotation matrix
        """
        q = np.array(quaternion, dtype=np.float64, copy=True)
        nq = np.dot(q, q)
        eps = np.finfo(float).eps * 4.0
        assert nq > eps
        q *= np.sqrt(2.0 / nq)
        q = np.outer(q, q)
        return np.array((
            (1.0 - q[1, 1] - q[2, 2], q[0, 1] - q[2, 3], q[0, 2] + q[1, 3]),
            (q[0, 1] + q[2, 3], 1.0 - q[0, 0] - q[2, 2], q[1, 2] - q[0, 3]),
            (q[0, 2] - q[1, 3], q[1, 2] + q[0, 3], 1.0 - q[0, 0] - q[1, 1])
        ), dtype=np.float64)

if __name__ == '__main__':
    main()
```

Бонус

Желающим отдохнуть после выполнения задания предлагается прочитать небольшой рассказ по теме задания. Have fun.

***

“Фотография — это тайна о тайне. Чем больше она рассказывает Вам, тем меньше Вы знаете”

Диана Арбус (1923 — 1971)

Фотография. Вы никогда не задумывались, сколько всего она может запечатлеть и сколько всего может нам поведать? Любое ваше действие, шаг, ваши эмоции. По фотографии мы можем судить не только о том, что запечатлено на кадре, но и о том, кто его создал. Вы можете узнать те руки, что делали в тот момент фото, зная всё про фотографию как отдельный вид искусства, как великое чудо, совершившееся за всю историю человечества. Я знаю про фотографии всё. Каждая деталь фото имеет значение. Знаете, что самое важное в фото? Думаю, нет. Ракурс. Возможно, профессиональным фотографам покажется, что я говорю глупости, но я – не профессионал. Если говорить о социальном значении фото, то ракурс – это самое главное. У медали две стороны, так, кажется, говорят? Так вот, у нее всего две стороны. На самом же деле, бывает и больше двух. Ты не сможешь знать все про объект, взглянув на него только с одного ракурса, в перспективе одного горизонта.

— На этом ваше драматичное выступление окончено, и мы можем перейти к делу?

— Как пожелаете. Я изучил фото, которое вы мне дали. В который раз полиция обращается ко мне. Я не детектив, даже не следователь. Фотографии – мое хобби, не более.

— Бросьте скромничать. Мы обращаемся к вам за помощью только в самых крайних случаях. Преступник скрылся, фотограф не найден, есть только это простое фото. На первый взгляд, и правда кажется, что это обыкновенное фото, человек со спины. Человек в пиджаке и в шляпе идет по улице. Просто идет.

— Это не простое фото, как и любое другое, знаете ли. Но взгляните на самое главное, о чем я вам только что говорил – ракурс.

— Фото сделано со спины, я не слепой – чуть обиженно сказал работник полиции.

— Нет, нет. Его сделал явно не идущий за будущей жертвой прохожий, его сделал человек, будто лежащий на земле или же маленький ребенок, взгляните же.

— И правда. Но фото сделано явно не ребенком, фотоаппарат профессиональный. Ребенок не справился бы с таким.

— Соглашусь с вами, что это не ребенок. Но дело вовсе не в фотоаппарате, который, кстати говоря, ни секунды не профессиональный. Дело в другом. Я с детства увлекаюсь фотографиями, и могу сказать – дети никогда не наводят курсив фотоаппарата наверх, если там конечно не мама или какой-то другой занимательный предмет. Здесь фотограф явно хотел остаться незамеченным или же устремить наше (может быть вовсе и не на наше) внимание на небо и правую часть фото. Ибо сам человек здесь будто бы и не играет особой роли. Ах да, извините, через пару минут этого человека беспощадно убили, оставив на месте преступления это фото, сделанное те же самые пару минут назад, судя по качеству фото. Итак, если это не ребенок, то кто? Есть предположения?

— Хм. Человек, сидящий или лежащий на дороге. Но это же бред.

— Зачем кому-то лежать на дороге? Это довольно-таки странно, не находите?

— Послушайте, если вы не знаете, при каких обстоятельствах сделано фото, так и скажите, мне кажется, что я зря теряю время.

— Что ж, я хотел подвести вас к ответу, но раз уж вы не хотите терять ваше драгоценное время... Я попробую вкратце объяснить, при условии, что вы не будете меня перебивать. Знаете такой шпионский прием: «хотите осмотреться вокруг, не вызывая подозрений? Присядьте и завяжите

шнурки!»? Кадр сделан слегка наклоненным вправо, другой рукой фотограф как бы завязывал шнурки, а другой делал фото. Делаем вывод, фотограф левша, так как завязывать шнурки удобнее той рукой, которой пишешь. Но зачем ему надо было осмотреться? Он шпион, или ему кажется, что за ним шпионят? Знаете, у каждой профессии свой стиль фотографирования. Врач хочет уместить в кадр как можно больше частей тела, художнику важен пейзаж, а знаете, что предпочитает в свое фото актер? Движение. Идущий человек, параллельно размахивающий руками, ну, не идеальный кадр для нашего актера? И все же ракурс тут самое важное. То, с какого положения сделан кадр, все решит. У вас есть подозреваемые?

— Да, семеро. Кстати, один из них актер... Брат убитого. Это он? Могу рассказать о его мотивах...

— Нет, нет. Меня не интересуют мотивы, как я уже говорил, я не детектив. Я могу взглянуть на этого человека и провести с ним кое-какие эксперименты?

— Разумеется. Поехали в участок, он там. Перед следователем и Фотографом сидел молодой человек.

— Покиньте нас на пару минут, – обратился Фотограф к следователю.

Следователь покорно вышел. Фотограф достал из кармана куртки небольшой фотоаппарат и вручил несчастному актеру.

— Что мне прикажете сделать? Фотографировать на фоне этих чудесных сцен?

— Если хотите, конечно. Возьмите его двумя руками и представьте, что вы действительно фотографируете меня.

Парень покорно сделал то, о чем его попросили. Затем Фотограф аккуратно забрал фотоаппарат обратно к себе.

— Ну что же. Тот кадр, который нашли, действительно сделали вы. Не знаю ваших целей и мотивов, это уже не моя работа, но кадр точно ваш.

— Послушайте меня, умоляю. Мой брат – ужасный человек. Я следил за ним с этим полароидом, черт его побрал. Хотел показать его жене, куда ходит ее любимый муж...

— Мне это неинтересно. Советую рассказать все следователю, пока не поздно. Всего доброго, до свидания.

Фотограф ушел. Тут же на него накинулся с вопросами следователь.

— Ну что, что? Вы выяснили все, что нужно?

— Да, вполне. Молодой человек вам все расскажет, кадр сделал он же.

— Но как?..

— Руки, друг мой, руки! Ладно, не думаю, что вам дано знать все мои секреты и уловки. Всего хорошего!

С этими словами Фотограф покинул полицейский участок, а следователь заворожённо смотрел ему вслед. Возможно, фотография это и не такая ерунда, как он считал раньше?..