Basic Camera Tracker, Part 1

Дедлайн — 04.11

# Задача

С помощью библиотеки OpenCV и выданного преподавателями дополнительного кода реализовать построение облака точек с помощью триангуляции и трекинг камеры путем решения задачи P*n*P.

# Детали

* Основная цель задания — получить на выходе последовательность позиций камеры и облако 3D-точек в мировом пространстве.
* Всевозможные константы, необходимые для настройки методов трекинга, подобрать самостоятельно.
* Домашнее задание нужно выполнять с использованием [выдаваемого нами кода](https://github.com/cvcourse/home-assignments-autumn-2022):
  + Реализацию трекинга нужно писать в файле camtrack.py. В нем есть какой-то минимальный код, его можно менять весь, но так, чтобы не изменить интерфейс модуля camtrack.
  + Файл \_camtrack.py содержит вспомогательный код, но реализованный преподавателями. Прежде чем что-то писать самостоятельно, убедитесь, что это еще не сделали за вас.
  + Если очень хочется или нужно, разрешается менять код в файле \_camtrack.py, но так, чтобы не сломать уже существующие интерфейсы.
  + Файл camtrack.py может использоваться как консольное приложение для запуска вашей реализации трекинга и отображения ошибок репроекции.
  + Тестировать результаты трекинга можно с помощью скрипта cmptrack.py.
  + Прогнать ваше решение сразу на всём тестовом наборе домашнего задания можно с помощью скрипта testrunner.py.
  + Визуализировать результаты трекинга можно с помощью render.py.

# Требования

* Ваш трекер должен выводить в stdout информацию о прогрессе трекинга: какой кадр обрабатывается; число инлаеров, по которым найдена позиция; число триангулированных на текущем этапе точек; размер облака точек в данный момент и т. п.
* Необходимо реализовать (и использовать) ретриангуляцию точек: уточнять позиции 3D-точек для достаточно длинных треков в процессе работы алгоритма. Ретриангуляция точки должна проводиться по более чем двум кадрам.
* Алгоритм должен работать пока есть хотя бы один кадр с неизвестной позицией, для которого есть необходимое количество 2D-3D соответствий. Последовательное отслеживание кадров не всегда позволяет это сделать.
* Ваше решение должно корректно завершаться для любого теста из набора тестов домашнего задания, перечисленных в файле dataset\_ha2.yml.
* Для теста с простым и медленным движением bike\_translation\_slow медиана ошибки вращения должна быть в пределах двух градусов, а медиана ошибки параллельного переноса должна быть в пределах 0.1 от длины трека.
* Для всех тестов максимальная ошибка по вращению должна быть не больше 30°, а максимальная ошибка по параллельному переносу не больше 0.3.

# Подсказки

## Общий алгоритм

* Обратите внимание: начальные кадры с известной позицией могут быть любыми, например:
  + Два соседних кадра.
  + Первый и последний кадр.
* Как может выглядеть базовый алгоритм:

1. Дополнить облако точек с учетом известных положений камер (два исходных в начале).
2. Выбрать кадр с неизвестным положением, для которого можно (лучше всего) решать P*n*P.
3. Определить положение камеры в выбранном кадре.
4. Повторять 1–3 пока есть кадры для пункта 2.

## Триангуляция и трекинг

* Проверяйте угол триангуляции и ошибки репроекции при вычислении позиций 3D-точек.
* Используйте RANSAC при решении задачи P*n*P.
* Уточняйте решение P*n*P на инлаерах путем итеративной минимизации ошибки репроекции. Реализация есть в OpenCV, но она без M-оценок (см. доп. задание 1).
* Для добавления 3D-точки нужно найти пару кадров, на которых будет достаточно большой угол триангуляции и маленькая ошибка репроекции.
* В процессе трекинга фильтруйте выбросы.
* Используйте 2D-метрики качества для фильтрации соответствий.
* Пересчет уже известных позиций камеры после ретриангуляции может улучшить точность решения.

## Тестирование

* Набор данных для тестирования трекинга расположен на Google Drive в директории курса.
* Описания тестов можно найти в файле dataset\_ha2.yml.
* Для сравнения результатов трекинга с ground truth используйте cmptrack.py.
* Прогоняйте ваше решение на всем датасете с помощью testrunner.py.
* Используйте camtrack.py --show для визуализации ошибок репроекции.
* Обратите внимание, что corners.py позволяет отследить уголки и сохранить их на диск, а camtrack.py позволяет считывать уголки с диска. Пользуйтесь этим, чтобы не тратить время на пересчет уголков при отладке.
* Смотрите на результаты трекинга глазами с помощью render.py.
* Для тестов fox\_head\_short и ironman\_translation\_fast можно найти подсчитанные преподавателями уголки. В процессе тестирования можете попробовать использовать их и сравнить результаты.

## OpenCV

* Обратите внимание: в OpenCV под термином *camera pose* обычно понимается видовая трансформация, т. е. обратная трансформации, задающей положение камеры в мировом пространстве.

# Дополнительные задания

1. **(3 балла) P*n*P c M-оценками**Реализовать робастный метод решения задачи P*n*P путем итеративной минимизации ошибки репроекции с M-оценками.