Руководство разработчика OrthoStitcher.v1.0

Дата: Май 2025
Версия: 1.0

---

1. Введение

Данное руководство предназначено для программистов и разработчиков, которые планируют вносить изменения в исходный код программного средства OrthoStitcher.v1.0. Оно содержит детальное описание внутренней архитектуры, алгоритмов, правил кодирования, а также рекомендации по отладке и расширению функционала.

2. Структура Проекта

Проект `OrthoStitcher` организован модульным образом, что облегчает понимание и модификацию кода:

- `OrthoStitcher/` (корневая директория репозитория)
  - `OrthoScript.py` : Основной исполняемый скрипт, содержащий всю логику программы.
  - `requirements.txt` : Список всех Python-зависимостей проекта.
  - `README.md` : Главная страница репозитория с кратким описанием и инструкциями.
  - `LICENSE` : Файл с информацией о лицензии проекта.
  - `.gitignore` : Файл, определяющий, какие файлы и папки Git должен игнорировать.
  - `docs/` : Папка для полной документации (PDF-файлы, включая это руководство).
  - `test_data/` : Папка с небольшими примерами входных изображений для тестирования и демонстрации.
  - `screenshots/` : Папка с изображениями для `README.md` и демонстраций.
  - `venv_orthostitcher/` : (Создается локально) Виртуальное окружение Python.

Логические слои и модули программы (функции в `OrthoScript.py`):

- `main` **блок (** `if __name__ == "__main__":` **)**: Точка входа в программу. Отвечает за парсинг аргументов командной строки, загрузку списка изображений, инициализацию процесса сшивки и сохранение результата.

- `stitch_images(images, blend_width)` : Главная функция-оркестратор. Координирует последовательное объединение изображений, итеративно вызывая другие функции пайплайна.
- `detect_and_describe(image)` : Модуль для извлечения ключевых точек и их дескрипторов из изображений.
- `match_descriptors(des1, des2)` : Модуль для сопоставления дескрипторов двух изображений и фильтрации совпадений.
- `find_homography(kp1, kp2, matches)` : Модуль для вычисления матрицы гомографии на основе отфильтрованных совпадений.
- `combine_images(img1, img2, H, blend_width)` : Модуль, отвечающий за геометрическую трансформацию второго изображения, расчет нового холста и его компоновку с первым изображением.
- `blend_images(img1, img2, mask, blend_width)` : Модуль для выполнения градиентного смешивания в области перекрытия, обеспечивающий бесшовность.

3. Описание Ключевых Компонентов (Детально)

Каждая функция в `OrthoScript.py` имеет четко определенное назначение и хорошо задокументирована с использованием `docstrings` (как в этом файле). Данные между функциями передаются преимущественно в виде объектов `numpy.ndarray` (для изображений и матриц) и списков (`list`) объектов `cv2.KeyPoint` и `cv2.DMatch` .

- `detect_and_describe(image)` :
  - **Цель:** Найти инвариантные особенности на изображении.
  - **Реализация:** Использует `cv2.SIFT_create()` для инициализации SIFT-детектора. SIFT-дескрипторы являются 128-мерными векторами float32. Изображение конвертируется в оттенки серого перед обработкой. Разработчик может экспериментировать с параметрами SIFT при инициализации, такими как `nfeatures` , `nOctaveLayers` , `contrastThreshold` , `edgeThreshold` , `sigma` .
- `match_descriptors(des1, des2)` :
  - **Цель:** Сопоставить дескрипторы двух изображений и отфильтровать ложные соответствия.
  - **Реализация:** Применяет `cv2.BFMatcher()` для прямого сопоставления. Для фильтрации используется `knnMatch(k=2)` с последующим Lowe's Ratio Test (`m.distance < 0.75 * n.distance`), который эффективно отсеивает неоднозначные соответствия.
- `find_homography(kp1, kp2, matches)` :
  - **Цель:** Вычислить перспективную трансформацию между двумя изображениями.

    - **Реализация:** Использует `cv2.findHomography()` с методом `cv2.RANSAC`. Порог `50.0` определяет максимальное допустимое отклонение для точек, считающихся инлайерами. Для надежного вычисления требуется минимум 20 совпадений (этот порог задан в коде). Возвращает `None`, если совпадений недостаточно.
- `blend_images(img1, img2, mask, blend_width)`:
    - **Цель:** Выполнить градиентное смешивание для бесшовного перехода.
    - **Реализация:** Изображения временно преобразуются в `np.float32`. Использует `cv2.distanceTransform()` для генерации карты расстояний от границ `mask`, на основе которой вычисляется `alpha`-канал. Смешивание выполняется по формуле `img1 * (1 - alpha) + img2 * alpha`. Области вне маски сохраняют значения из `img1`.
- `combine_images(img1, img2, H, blend_width)`:
    - **Цель:** Объединить два изображения на новом холсте с учетом трансформации и смешивания.
    - **Реализация:** Вычисляет размеры нового холста, достаточные для размещения обоих изображений. Создает матрицу сдвига (`translation`) для корректного позиционирования. Использует `cv2.warpPerspective()` для трансформации второго изображения и его маски. К маске применяется морфологическая операция эрозии (`cv2.erode`) для улучшения качества смешивания. Затем вызывает `blend_images`.

4. Правила Кодирования и Документирования

Для поддержания высокого качества кода и облегчения командной работы рекомендуется придерживаться следующих правил:

- **Стандарт стиля кода:** Соответствие рекомендациям PEP 8 по форматированию кода.
- **Именование:** Использовать `snake_case` для имен функций, переменных и файлов (например, `detect_and_describe`, `blend_width`, `OrthoScript.py`).
- **Документирование функций (Docstrings):** Каждая функция должна иметь подробный `docstring` в формате reStructuredText (или аналогичном, как в этом файле), описывающий ее назначение, входные параметры (`Args:`) с указанием типов и возвращаемые значения (`Returns:`).
- **Встроенные комментарии (Inline comments):** Использовать `#` для пояснения сложных, неочевидных или алгоритмически важных участков кода.

5. Отладка Программы

Для эффективной отладки `OrthoStitcher` рекомендуется использовать следующие методы:

- **IDE-отладчики:** Используйте встроенные отладчики PyCharm или Visual Studio Code. Они позволяют устанавливать точки останова (`breakpoints`), пошагово выполнять код, просматривать значения переменных в реальном времени.
- **Вывод в консоль (`print()`):** Для быстрого контроля значений переменных или хода выполнения можно временно добавлять операторы `print()` . Удаляйте или комментируйте их после завершения отладки.
- **Визуализация промежуточных результатов (`cv2.imshow()`):** Это мощный инструмент для визуальной отладки. Используйте `cv2.imshow()` для отображения изображений на различных этапах обработки (например, после детектирования ключевых точек, после трансформации, после смешивания). Не забудьте `cv2.waitKey(0)` и `cv2.destroyAllWindows()` для управления окнами.

```
cv2.imshow("Промежуточное изображение", image_array)
cv2.waitKey(0) # Ожидать нажатия любой клавиши
cv2.destroyAllWindows() # Закрыть все окна OpenCV
```

- **Типичные точки останова для отладки:**
  - После `match_descriptors` : Проверить количество и качество найденных совпадений.
  - После `find_homography` : Проверить содержимое матрицы `H` и её корректность.
  - Внутри `combine_images` после `cv2.warpPerspective` : Визуализировать трансформированное второе изображение.
  - Внутри `blend_images` : Проверить, как формируется `alpha` -канал и выглядит результат смешивания.

6. Тестирование

Разработчик должен проводить тестирование своих изменений для обеспечения стабильности и корректности работы Программы.

- **Ручное тестирование:** Запуск `OrthoScript.py` с различными тестовыми наборами данных из папки `test_data` . Включайте как "идеальные" случаи, так и "краевые" (например, низкое перекрытие, изменения освещения, шумные изображения). Визуальная оценка качества выходной панорамы остается ключевым методом.
- **Автоматизированные тесты (рекомендуется):** Настоятельно рекомендуется разрабатывать модульные и интеграционные тесты с использованием стандартных фреймворков, таких как `unittest` (встроенный в Python) или `pytest` .
  - **Модульные тесты:** Проверка корректности работы отдельных функций изолированно (например, `detect_and_describe` возвращает ожидаемый формат данных, `match_descriptors` корректно фильтрует совпадения).

- **Интеграционные тесты:** Проверка взаимодействия между функциями и всего пайплайна (например, сшивка небольшого тестового набора изображений и сравнение результата с эталонным).

7. Расширение и Модификация Функционала

Архитектура программы спроектирована с учетом возможности расширения функционала и замены отдельных компонентов.

- **7.1. Основные точки расширения:**
  - **Альтернативные алгоритмы детектирования/описания признаков:**
    - Модификация функции `detect_and_describe` для поддержки других классических алгоритмов (например, ORB, AKAZE), а также интеграция современных нейросетевых методов, таких как **SuperPoint**.
  - **Сопоставление признаков на основе глубокого обучения:**
    - Интеграция **SuperGlue** для сопоставления признаков, что может значительно повысить точность и устойчивость к сложным сценариям, а также **LoFTR (Local Feature Transformer)** для прямого предсказания плотных соответствий.
  - **Усовершенствованные методы смешивания:**
    - Реализация более сложных алгоритмов смешивания в `blend_images`, таких как многополосное смешивание (multi-band blending), для достижения ещё более высокого качества и устранения остаточных артефактов при значительных различиях в освещении.
  - **Оптимизация производительности:**
    - Исследование и внедрение параллельных вычислений на уровне CPU (например, с использованием модуля `multiprocessing` для обработки независимых пар изображений).
    - Исследование возможностей GPU-ускорения (например, через `opencv-python-cuda` и библиотеки CUDA) для нейросетевых методов.
  - **Переход к 3D-моделированию и ортотрансформированию:**
    - Развитие программы в полноценную фотограмметрическую систему, включающую этапы создания трехмерных моделей. Это потребует:
      - **Расчет 3D-структуры:** "Поднятие" 2D-соответствий в 3D-пространство (например, с использованием алгоритмов Structure from Motion - SfM, или методов **LoFT** - Lifting from the 2D to 3D).
      - **Построение Цифровой Модели Рельефа (ЦМР/DEM):** Создание цифрового представления поверхности Земли.
      - **Использование RANSAC 3D:** Применение алгоритма RANSAC для вычисления трансформаций или моделей в 3D-пространстве (например, для совмещения 3D-моделей или фильтрации выбросов в облаках точек).

        - **Ортотрансформирование:** Проецирование изображений на ЦМР для устранения перспективных искажений, вызванных рельефом местности.
    - **Парсинг и использование референтных данных:**
        - Добавление функционала для парсинга и использования спутниковых или картографических изображений из открытых источников (например, Google Maps, данные Leaflet) в качестве референтных слоев для геопривязки, валидации или контекстуализации.
    - **Разработка графического интерфейса пользователя (GUI):**
        - Создание удобного и интуитивно понятного пользовательского интерфейса с использованием библиотек, таких как PyQt, Tkinter или Streamlit.

- **7.2. Принципы добавления нового функционала:**
    - **Модульность:** Новый функционал следует реализовывать в виде отдельных функций или классов, которые затем интегрируются в существующий пайплайн (`stitch_images` или `combine_images`).
    - **Параметризация:** По возможности, новый функционал должен быть конфигурируем через аргументы командной строки, аналогично `blend_width` и `direction`.
    - **Совместимость:** Новые функции должны поддерживать текущие форматы входных и выходных данных, или должна быть предусмотрена конвертация.

- **7.3. Области для рефакторинга и улучшения:**
    - **Обработка ошибок:** Внедрение централизованной системы логирования (с использованием модуля `logging`) и/или пользовательских исключений для более сложной отладки и расширяемости.
    - **Структурирование `combine_images`:** Разделение этой объемной функции на несколько более мелких и специализированных подфункций (например, `calculate_canvas_size`, `warp_image_and_mask`, `place_first_image`) для повышения читаемости и поддерживаемости кода.
