ФГБОУ ВО «Уфимский университет науки и технологий»

Институт информатики, математики и робототехники

Кафедра высокопроизводительных вычислений и дифференциальных уравнений

**ОТЧЕТ О ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ**

**ТИП ПРАКТИКИ**

научно-исследовательская работа

**ОБУЧАЮЩЕГОСЯ**

4 курса группы ПМ-457

Акмурзина Михаила Эдуардовича

|  |  |
| --- | --- |
| Уровень образования: | высшее образование – бакалавриат |
| Направление подготовки (специальность) | 01.03.04 Прикладная математика |
| Направленность (профиль) программы | Применение математических методов к решению инженерных и экономических задач |
| Срок проведения практики: | с 23 декабря 2024 по 25 января 2025 |

**1. МЕТОДИЧЕСКИЕ УКАЗАНИЯ**

1. База практики – профильная организация или структурное подразделение УУНиТ.
2. Обучающийся – физическое лицо, осваивающее образовательную программу среднего профессионального или высшего образования.
3. Вид практики – учебная, производственная.
4. Каждый обучающийся, находящийся на практике, обязан вести отчет по практике.
5. Отчет по практике служит основным и необходимым материалом для составления обучающимся отчета о своей работе на базе практики.
6. Заполнение отчета по практике производится регулярно, аккуратно и является средством самоконтроля. Отчет можно заполнять рукописным и (или) машинописным способами.
7. Иллюстративный материал (чертежи, схемы, тексты и т.п.), а также выписки из инструкций, правил и других материалов могут быть выполнены на отдельных листах и приложены к отчету.
8. Записи в отчете о практике должны производиться в соответствии с программой по конкретному виду практики.
9. После окончания практики обучающийся должен подписать отчет у руководителя практики, руководителя от базы практики и сдать свой отчет по практике вместе с приложениями (при наличии) на кафедру.
10. При отсутствии сведений в соответствующих строках ставится прочерк.

**2. ОБЩИЕ ПОЛОЖЕНИЯ**

|  |  |
| --- | --- |
| Фамилия, инициалы, должность руководителя практики от факультета (института, колледжа, техникума) | ⸺ |
| Фамилия, инициалы, должность руководителя практики от кафедры | доцент каф. ВВиДУ Лукащук В.О. |
| Полное наименование базы практики | ФГБОУ ВО «Уфимский университет науки и технологий» |
| Наименование структурного подразделения базы практики | кафедра высокопроизводительных вычислений и дифференциальных уравнений |
| Адрес базы практики (индекс, субъект РФ, район, населенный пункт, улица, дом, офис) | 450008, респ. Башкортостан, г. Уфа, ул. К.Маркса, 12к1, ауд. 1-407 |
| Фамилия, инициалы, должность руководителя практики от профильной организации | доцент каф. ВВиДУ Касаткин А.А. |
| Телефон руководителя практики от базы практики | ⸺ |

**3. РАБОЧИЙ ГРАФИК (ПЛАН) ПРОВЕДЕНИЯ ПРАКТИКИ**

Срок проведения практики: с 23 декабря 2024 по 25 января 2025

|  |  |  |  |
| --- | --- | --- | --- |
| № | Разделы (этапы) практики | Виды и содержание работ, в т.ч. самостоятельная работа обучающегося в соответствии с программой практики | График (план) проведения практики  (начало – окончание) |
| 1. | Подготовительный этап | – организационное собрание;  – установочная лекция;  – получение индивидуального задания на практику;  – проведение инструктажа обучающегося по ознакомлению с требованиями охраны труда, техники безопасности, пожарной безопасности, а также правилами внутреннего трудового распорядка. | 23.12.2024 – 25.12.2024 |
| 2. | Основной этап | – выполнение индивидуального задания;  – сбор, обработка и систематизация фактического и литературного материала по теме исследования. | 26.12.2024 – 21.01.2025 |
| 3. | Заключительный этап | – подготовка и оформление отчёта по практике, содержащего итоги прохождения практики;  – подготовка к защите, в том числе оформление презентации, и защита отчета. | 22.01.2025 – 25.01.2025 |

|  |  |
| --- | --- |
| Руководитель практики от кафедры | \_\_\_\_\_\_\_\_\_\_\_\_/ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись И.О. Фамилия |
| Руководитель практики от профильной организации [[1]](#footnote-1) | \_\_\_\_\_\_\_\_\_\_\_\_/ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись И.О. Фамилия |

**4. ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ**

Содержание задания на практику (перечень подлежащих рассмотрению вопросов, выполняемых работ, связанных с будущей профессиональной деятельностью):

1. Разработать инструмент для разметки изображений с дорожной камеры видеонаблюдения с указанием геометрических ограничений (например, параллельность прямых)
2. Разработать и реализовать алгоритм определения параметров в матрице проекции, обеспечивающих наилучшее выполнение заданных геометрических ограничений
3. Проанализировать и сравнить результаты расчетов на примере реальных и синтетических данных.

|  |  |
| --- | --- |
| Руководитель практики от кафедры | \_\_\_\_\_\_\_\_\_\_\_\_/ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись И.О. Фамилия |
| Руководитель практики от профильной организации | \_\_\_\_\_\_\_\_\_\_\_\_/ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись И.О. Фамилия |
| ОЗНАКОМЛЕН:  Обучающийся | \_\_\_\_\_\_\_\_\_\_\_\_/ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись И.О. Фамилия |

**5. ИНСТРУКТАЖ ПО ОХРАНЕ ТРУДА**

Наименование и реквизиты локального нормативного акта, регламентирующего систему управления охраной труда, техники безопасности, пожарной безопасности профильной организации:

Положение о системе управления охраной труда в ФГБОУ ВО «УУНиТ», утвержденное приказом №0632 от 20.03.2023.

Инструкция по охране труда для неэлектротехнического персонала I квалификационной группы допуска по электробезопасности (ИОТ-УУНиТ-002-2023) от 01.02.2023.

Инструкция по охране труда «Организация безопасного передвижения по лестницам в образовательной организации» (ИОТ-СОТ-004-2023) от 16.01.2023.

Инструкция о мерах безопасности при эвакуации работников и обучающихся УУНиТ при пожаре, утвержденная приказом УУНиТ №710 от 26.12.2022.

Наименование и реквизиты локального нормативного акта, устанавливающего правила внутреннего трудового распорядка профильной организации:

Правила внутреннего трудового распорядка Уфимского университета науки и технологий, утвержденные приказом УУНиТ “Об утверждении Правил внутреннего трудового распорядка федерального государственного бюджетного образовательного учреждения высшего образования «Уфимский университет науки и технологий»” №0171 от 30.01.2023.

Правила внутреннего трудового распорядка обучающихся в Уфимском университете науки и технологий, утвержденные приказом УУНиТ от 23.05.2023 №1285 " Об утверждении Правил внутреннего распорядка обучающихся".

Перед началом практики инструктаж по ознакомлению с требованиями охраны труда, техники безопасности, пожарной безопасности, а также правилами внутреннего трудового распорядка прошел:

обучающийся \_\_\_\_\_\_\_\_\_\_\_\_/ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись И.О. Фамилия

Перед началом практики инструктаж обучающегося по ознакомлению с требованиями охраны труда, техники безопасности, пожарной безопасности, а также правилами внутреннего трудового распорядка провел:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_/ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

должность подпись И.О. Фамилия

**6. ДНЕВНИК РАБОТЫ ОБУЧАЮЩЕГОСЯ**

|  |  |
| --- | --- |
| Дата | Информация о проделанной работе, использованные источники и литература (при наличии) |
| 23.12.2024 -25.12.2024 | Согласование задания с научным руководителем |
| 25.12.2024 -28.01.2025 | Изучение литературы |
| 9.01.2025 –  12.01.2025 | Разработка инструмента для разметки изображений с дорожной камеры видеонаблюдения с указанием геометрических ограничений |
| 21.01.2025 – 22.01.2025 | Разработка и реализация алгоритма определения параметров в матрице проекции. |
| 23.01.2025-24.01.2025 | Оформление отчета. Подготовка к защите результатов практики |
| Руководитель практики от кафедры | \_\_\_\_\_\_\_\_\_\_\_\_/ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись И.О. Фамилия |
| Руководитель практики от профильной организации | \_\_\_\_\_\_\_\_\_\_\_\_/ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись И.О. Фамилия |

**7. ОТЧЕТ ОБУЧАЮЩЕГОСЯ О ПРАКТИКЕ**

Я, Акмурзин Михаил Эдуардович, прошел производственную (преддипломную) практику с 23 декабря 2024 по 25 января 2025

В соответствии с программой практики и индивидуальным заданием я выполнял следующую работу:

1. Разработал инструмент для разметки изображений с дорожной камеры видеонаблюдения с указанием геометрических ограничений (например, параллельность прямых)
2. Разработал и реализовать алгоритм определения параметров в матрице проекции, обеспечивающих наилучшее выполнение заданных геометрических ограничений
3. Проанализировал и сравнил результаты расчетов на примере реальных и синтетических данных.

В результате прохождения практики поставленные задачи были решены в полном объеме, профессиональные компетенции (профессиональные умения, навыки и опыт профессиональной деятельности) приобретены.

|  |  |
| --- | --- |
| Обучающийся | \_\_\_\_\_\_\_\_\_\_\_\_/ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись И.О. Фамилия |

**8. ЗАКЛЮЧЕНИЕ РУКОВОДИТЕЛЯ ПО ПРАКТИЧЕСКОЙ ПОДГОТОВКЕ О ПРАКТИКЕ**

Обучающийся Акмурзин Михаил Эдуардович прошел производственную (преддипломную) практику с 23 декабря 2024 по 25 января 2025.

Перед обучающимся во время прохождения практики были поставлены следующие профессиональные задачи:

1. Разработать инструмент для разметки изображений с дорожной камеры видеонаблюдения с указанием геометрических ограничений (например, параллельность прямых)
2. Разработать и реализовать алгоритм определения параметров в матрице проекции, обеспечивающих наилучшее выполнение заданных геометрических ограничений
3. Проанализировать и сравнить результаты расчетов на примере реальных и синтетических данных.

Краткая характеристика проделанной работы и полученных результатов:

Во время прохождения практики обучающийся проявил себя как самостоятельный исследователь, способный выполнять.

Рекомендации (пожелания) по организации практики: нет

|  |  |
| --- | --- |
| Руководитель практики от профильной организации | \_\_\_\_\_\_\_\_\_\_\_\_/ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  М.П. подпись И.О. Фамилия  «\_\_\_»\_\_\_\_\_\_\_\_\_20\_\_\_ |

**9. РЕЗУЛЬТАТ ЗАЩИТЫ ОТЧЕТА**

В результате прохождения практики поставленные задачи были решены в полном объеме, профессиональные компетенции (профессиональные умения, навыки и опыт профессиональной деятельности) приобретены.

Результат прохождения практики обучающимся оценивается на:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| Руководитель практики от кафедры | \_\_\_\_\_\_\_\_\_\_\_\_/ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись И.О. Фамилия |

**ПРИЛОЖЕНИЕ А  
(обязательное)**

**МОДЕЛЬ КАМЕРЫ ОБСКУРЫ**

Модель камеры-обскуры описывает математическую связь между координатами точки в трехмерном пространстве и ее проекцией на плоскость изображения идеальной камеры-обскуры, где апертура камеры описывается как точка, а линзы не используются для фокусировки света. Модель не включает, например, геометрические искажения или размытие несфокусированных объектов, вызванные линзами и апертурами конечного размера. Она также не принимает во внимание, что цифровые камеры имеют только дискретные координаты изображения. Это означает, что модель камеры-обскуры можно использовать только в качестве первого приближения преобразования 3D-сцены в 2D - изображение. Его достоверность зависит от качества камеры и, как правило, уменьшается от центра изображения к краям по мере увеличения эффектов искажения объектива. На рисунке 1 показана модель камеры-обскуры.

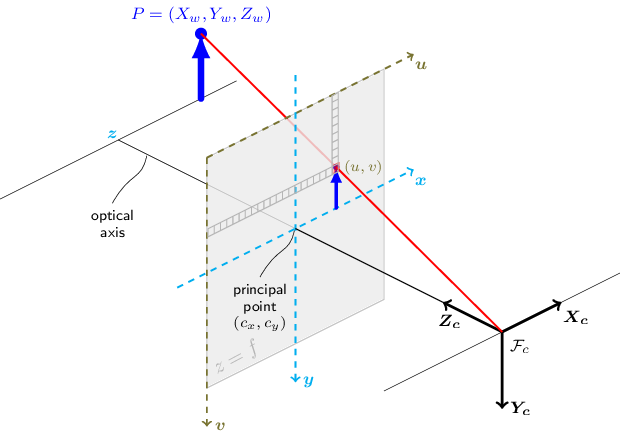


Рисунок 1 – Модель камеры-обскуры

Проективное преобразование, заданное моделью камеры-обскуры, показано ниже (1):

где – трехмерная точка в мировой системе координат (используются однородные координаты),

– двумерный пиксель в плоскости изображения (используются однородные координаты),

– внутренняя матрица камеры,

и – матрица поворота и вектор перемещения, описывающие изменение координат от мира к камере,

s – произвольное масштабирование проективного преобразования, не являющееся частью модели камеры.

Внутренняя матрица камеры проецирует 3D-точки, заданные в системе координат камеры, в 2D-пиксельные координаты то есть (2):

Элементы внутренней матрицы камеры (3) включают фокусные расстояния и , выраженные в пикселях, и сдвиг центральной точки (,), которая обычно находится близко к центру изображения:

Матрица внутренних параметров не зависит от просматриваемой сцены. Таким образом, после оценки её можно использовать повторно, если фокусное расстояние фиксировано (в случае зум-объектива). Таким образом, если изображение с камеры масштабируется с коэффициентом, все эти параметры необходимо масштабировать (соответственно умножать/делить) на один и тот же коэффициент.

Совместная матрица вращения-переноса из (1) является матричным произведением проективного преобразования и однородного преобразования. Проективное преобразование 3 на 4 (4) отображает 3D-точки, представленные в координатах камеры, в 2D-точки на плоскости изображения и представленные в нормализованных координатах камеры и :

Однородное преобразование определяется внешними параметрами и и представляет собой изменение базиса с мировой системы координат на систему координат камеры . Таким образом, учитывая представление точки в мировых координатах, , мы получаем представление P в системе координат камеры, , по формуле (5):

то есть матрица однородного преобразования состоит из – матрицы вращения, и  – вектора переноса:

Подставляя (3), (6) в (1) получим преобразование:

Для более компактного представления матрица поворота выразим ее через углы Тейта-Брайна [4]. Эти углы описывают последовательные вращения вокруг трех взаимно перпендикулярных осей, что позволяет представить матрицу поворота в компактной форме с помощью трех углов.

Для вращения вокруг оси мировой системы координат на угол :

Для вращения вокруг оси мировой системы координат на угол :

Для вращения вокруг оси мировой системы координат на угол :

Тогда перемножая матрицы (10), (11), (12) получим матрицу поворота, соответствующую последовательному вращению вокруг осей и зависящую от углов . При этом стоит отметить, что операция умножения матриц не является коммутативной, что означает, что порядок применения вращений имеет значение:

Раскрывая (13), получим:

Рассмотренная выше модель камеры обскуры (9) задает проекцию – точки в без учета искажений и имеет 10 степеней свободы (4 из внутренней матрицы, 6 из внешней матрицы). Однако в данной работе используется модифицированная модель [1], учитывающая ограничение сцены плоскостью дороги (. Она обладает 5ю степенями свободы (фокусное расстояние, углы матрицы вращения, высота камеры), что позволяет упростить задачу, при этом оставляя достаточно параметров для точной калибровки. Искажения объектива(дисторсия) корректируется нейронной сетью GeoCalib [3], предназначенную для автоматической калибровки камеры и коррекции искажений, возникающих в процессе съемки. Для ее описания скорректируем модель камеры обскуры:

где количество пикселей на изображение по горизонтали и вертикали соответственно,

– отношение количества пикселей по горизонтали к количеству пикселей по вертикали (15),

– двумерный пиксель в плоскости изображения (используются однородные координаты),

– точка в мировой системе координат в плоскости дороги (используются однородные координаты),

Отношения количества пикселей выражается:

где – количество пикселей по вертикале, – количество пикселей по горизонтали.

Составим обратное преобразование для уравнения (14) для перехода от координат изображения к реальным координатам. Для этого обозначим прямое преобразование

и получим:

**ГЕОМЕТРИЧЕСКИЕ ОГРАНИЧЕНИЯ СЦЕНЫ И ЦЕЛЕВАЯ ФУНКЦИЯ ОПТИМИЗАЦИИ**

Наложение геометрических ограничений на сцену наблюдения за дорожным движением позволяет значительно упростить задачу калибровки камеры [2]. В данном контексте можно использовать несколько ключевых геометрических примитивов, которые легко идентифицировать и применить к реальным условиям наблюдения. К таким примитивам относятся, например, параллельные прямые, нормали к ним, а также вычисление расстояний от точки до точки. Эти геометрические ограничения помогают в уточнении положения камеры и её ориентации. На рисунке 2 приведены примеры геометрических ограничений на сцену.

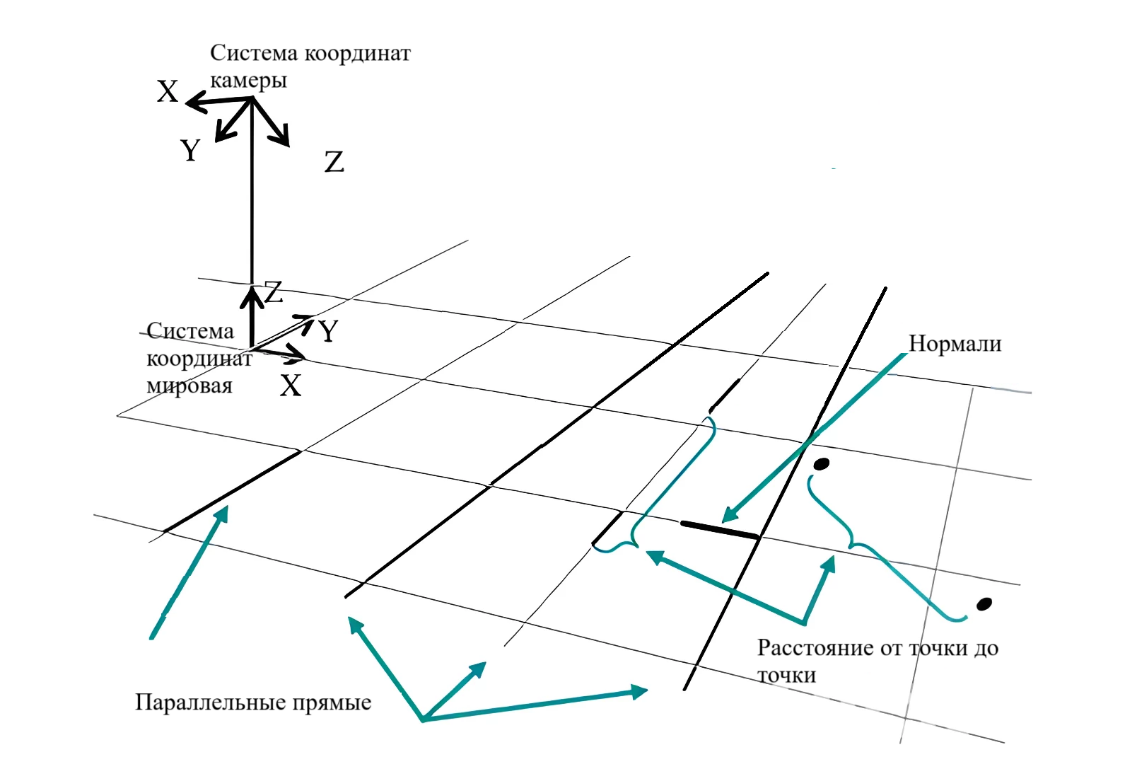


Рисунок 2 – Геометрические ограничения на сцене.

Рассмотрим, как эти ограничения могут быть определены и применены на практике:

1. Параллельные прямые

Одним из простых и часто встречающихся ограничений являются параллельные прямые. Эти прямые служат мощным инструментом для выравнивания сцены и определения её перспективы. В качестве таких прямых могут быть использованы линии, образующие бордюры или разметку дороги.

Рассмотрим набор параллельных линий , где такое что , где начало и конец отрезка на изображение. Спроецируем в мировые координаты точки по (17) и получим:

Вычитая из , получим вектор:

Вычисляя аналогично для прямой получим соответственно для двух векторов следующую функцию ошибки для параллельных прямых:

1. Нормали к прямым

В контексте дорожной сцены нормали могут использоваться для вычисления углов наклона, проверки параллельности линий или уточнения перспективных искажений. В качестве таких прямых могут быть использованы линии образующую разметку дороги.

Рассмотрим пары ортогональных линий , где такое что и , где начало и конец отрезка на изображение. Аналогично (18) и (19) спроецируем точки в мировые координаты и получим 2 вектора  . Следовательно функция ошибки для данного вида ограничения будет выглядеть:

1. Расстояние от точки до точки

Эти примитивные данные могут быть получены на основе знаний о структуре дороги (например, о разделении полос движения по продольной разметке, длине пешеходного перехода) или путем выполнения полевых измерений между ориентирами на местности. Это ограничение помогает определить масштаб сцены.

Рассмотрим следующую пару , где такое что , , где начало и конец отрезка на изображение. Аналогично (18) и (19) спроецируем точки в мировые координаты и получим вектор . Из этого следует, что функция ошибки для данного вида ограничения будет следующая:

Тогда совмещая все ограничения сцены, получим следующий целевой функционал:

где весовые коэффициенты, – вектор с параметрами камеры соответственно фокусное расстояние, углы вращения, высота.

Стоит отметить, что целевая функция может содержать несколько наборов ограничений одного типа. Важным моментом является то, что ограничения ортогональности и параллельности прямых могут быть взаимозаменяемыми.

Целевой функционал (23) решался с помощью МНК в Scipy [5].

**ПРИЛОЖЕНИЕ B  
(обязательное)**

**РЕЗУЛЬТАТЫ РАБОТЫ**

Приведя пример расчета на синтетическиx данных, создадим сцену, содержащую различные геометрические ограничения. На рисунке 3 изображен пример этой сцены в мировой системе координат. Сцена содержит 2 набора параллельных прямых, 2 пары ортогональных линий, 2 расстояние из точки в точку.

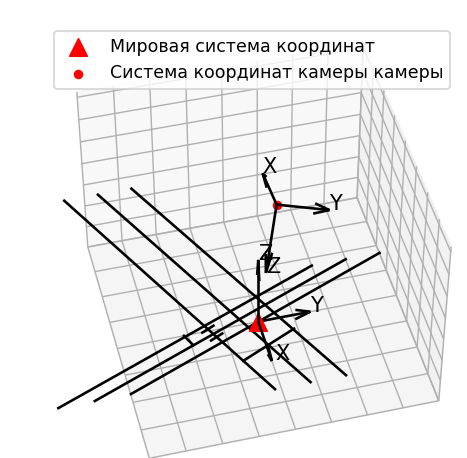


Рисунок 3 – Пример синтетической сцены

Зададим параметры камеры и спроецируем линии из мировой системы координат на плоскость изображения. Получим, вид сцены на рисунке 4. После чего воспользуемся оптимизацией для поиска параметров камеры и получим заданные значения с точностью 5.1028e-08. На рисунке 5 показан график сходимости целевого функционала.

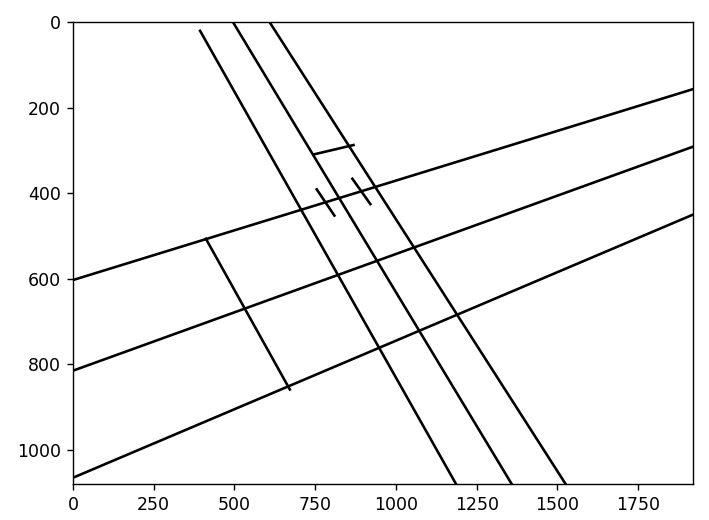


Рисунок 4 – Пример синтетической сцены на плоскости изображения

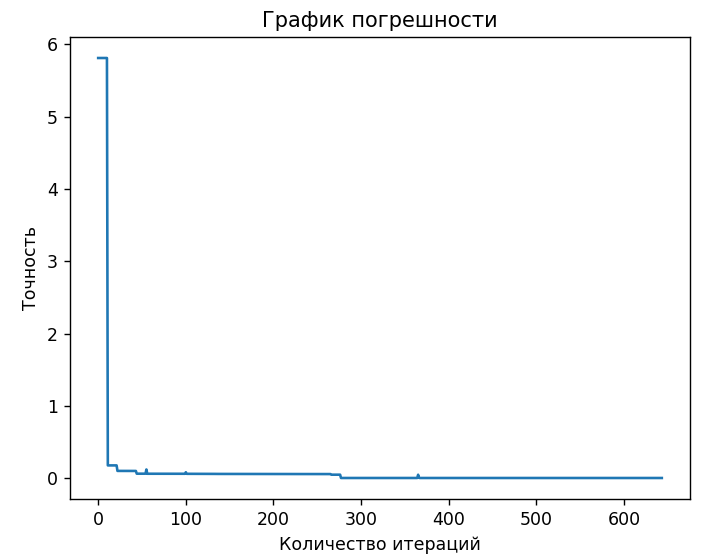


Рисунок 5 – График погрешности целевой функции

Приведем пример расчета на реальных данных. Для удобной разметки данных на изображении, создания и редактирования линий, а также сохранения результатов в файл, был создан инструмент с использованием библиотеки OpenCV [7]. Этот инструмент предназначен для того, чтобы пользователь мог удобно добавлять линии и точки на изображение, редактировать их, а затем сохранять разметку в файл для дальнейшего использования. Для его использования необходимо:

1. Загрузить изображение сцены в формате JPG/PNG
2. Кликая левой кнопкой мыши по изображению, выбрать один из видов ограничений на сцене. В случае редактировании, подвести стрелку на нужный край прямой, зажать левую кнопку мыши и переместить точку в нужное положение.
3. Для отмены действия нажать на правую кнопку мыши
4. Для сохранения результатов, зажать кнопку s, а для выхода Esc.

В качестве примера разметим перекресток Пушкина-Аксакова. На рисунке 6 приведен пример разметки параллельных прямых.



Рисунок 6 – Пример разметки данных через инструмент

На рисунке 7 изображены геометрические ограничения, используемые для калибровки камеры. Это соответственно два набора параллельных прямых и 11 расстояний от точки до точки. В качестве геометрического примитива расстояние от точки до точки был выбран пешеходный переход с длиной согласно ГОСТ Р 51256-2011 [6] от 4 до 6 метров, а для наборов параллельных линий соответственно полосы дороги.

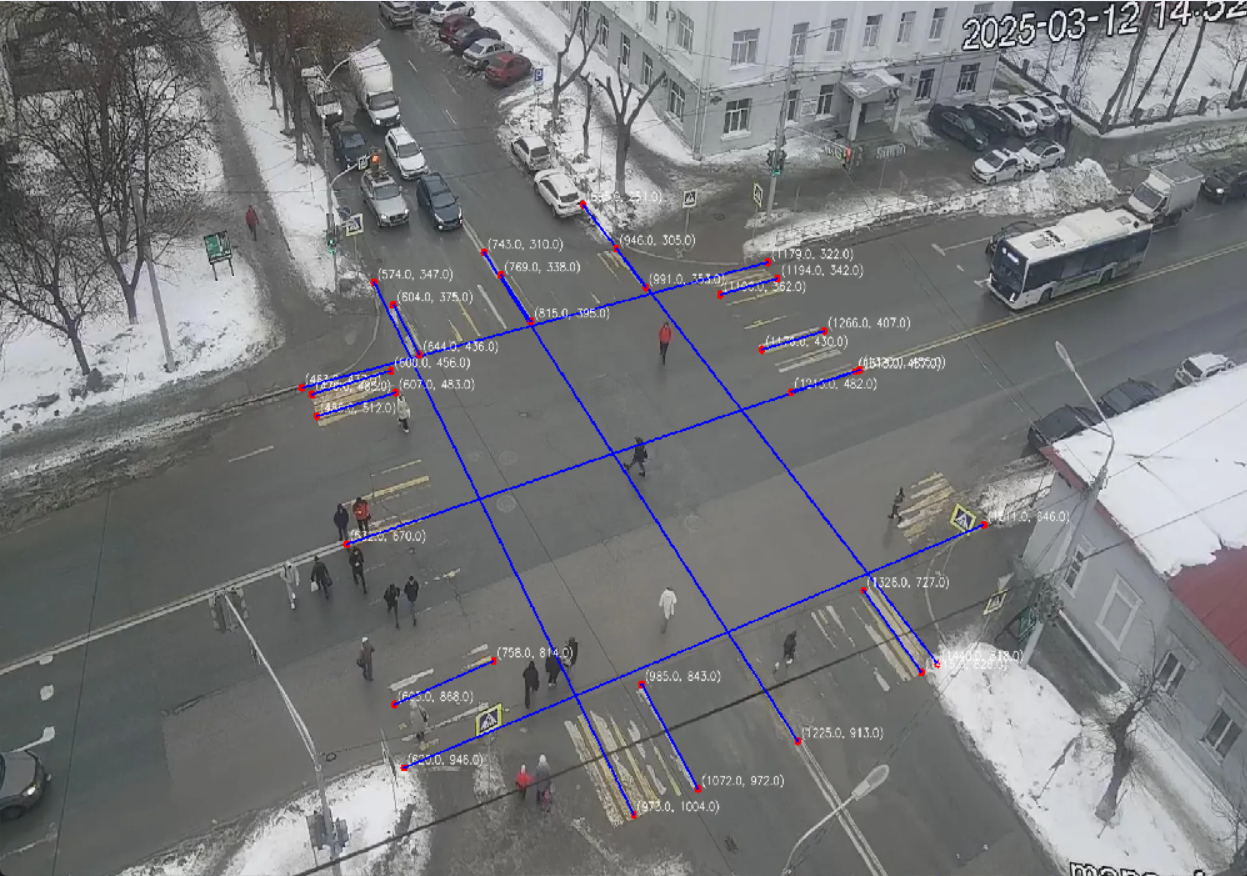
****

Рисунок 7 – Геометрические ограничения для калибровки камеры с перекрестка Пушкина – Аксакова

С использованием предоставленных наборов данных, произведем определение параметров камеры. Затем для проверки метода, выполнены два этапа проецирования: сначала из координат изображения в мировые координаты, а затем обратно из мировых координат в координаты изображения. На рисунке 8 показано, что исходные линии (жирные) практически совпадают с проецированными линиями (тонкие), что подтверждает точность выполненного проецирования.

Далее построим график сходимости целевой функции и погрешности для всех наборов данных. На рисунке 9 наглядно видно, что функция сходится с точностью 7.7165e-01 и при этом наибольшие отклонения наблюдаются для примитивов расстояний от точки до точки.

На основе вычисленных параметров камеры отобразим линии, соответствующие ширине полосы относительно центральных линий перекрестка. На рисунке 10 можно наблюдать, что наша модель отображает структуру полос и подходит для измерения скорости автомобиля, но также выявлена погрешность, вызванная искажениями объектива и шумом в наборах данных.

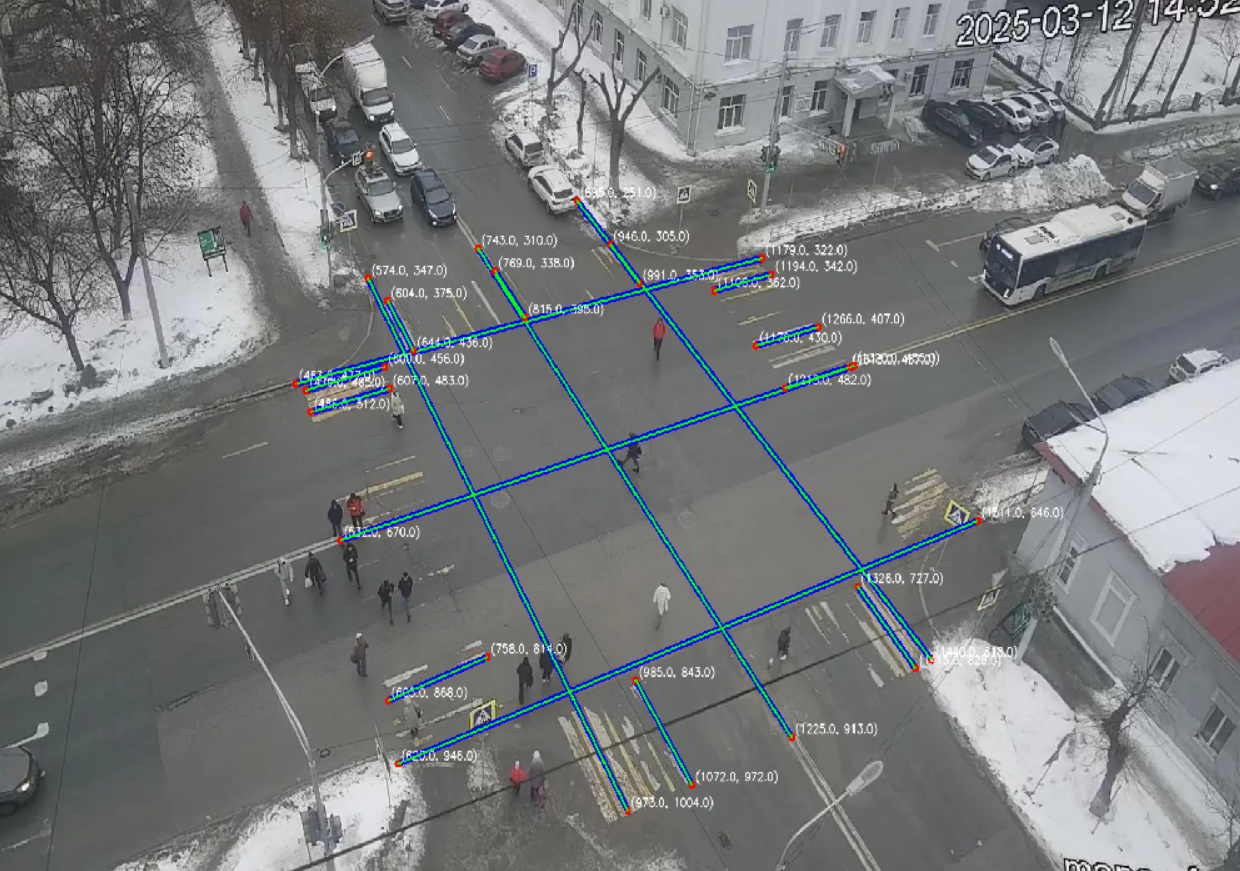
****

Рисунок 8 - Результат проецирования из координат изображения в мировые координаты и обратно.

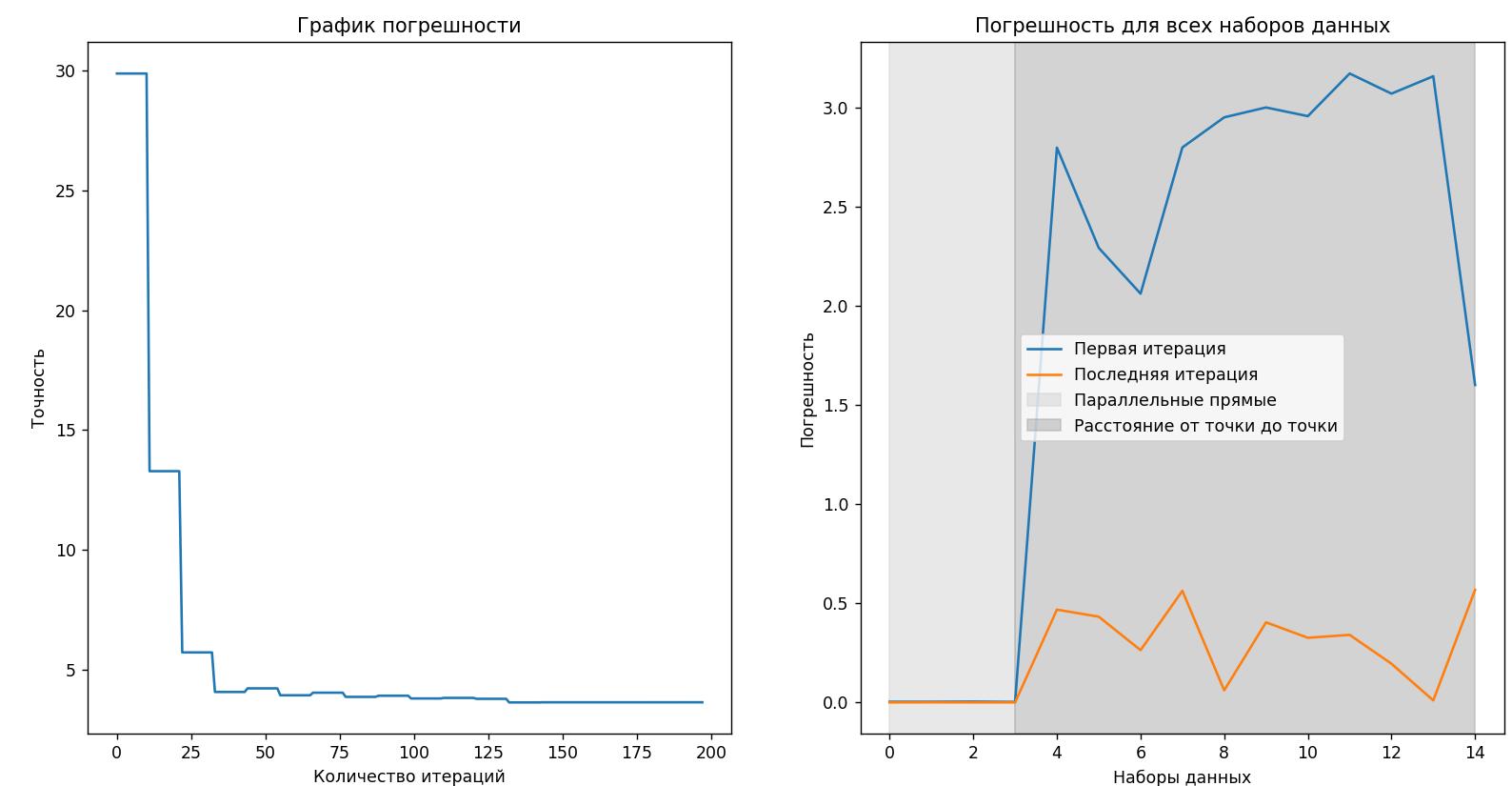


Рисунок 9 – График погрешности целевой функции и погрешности для всех наборов данных на первой и последней итерации.

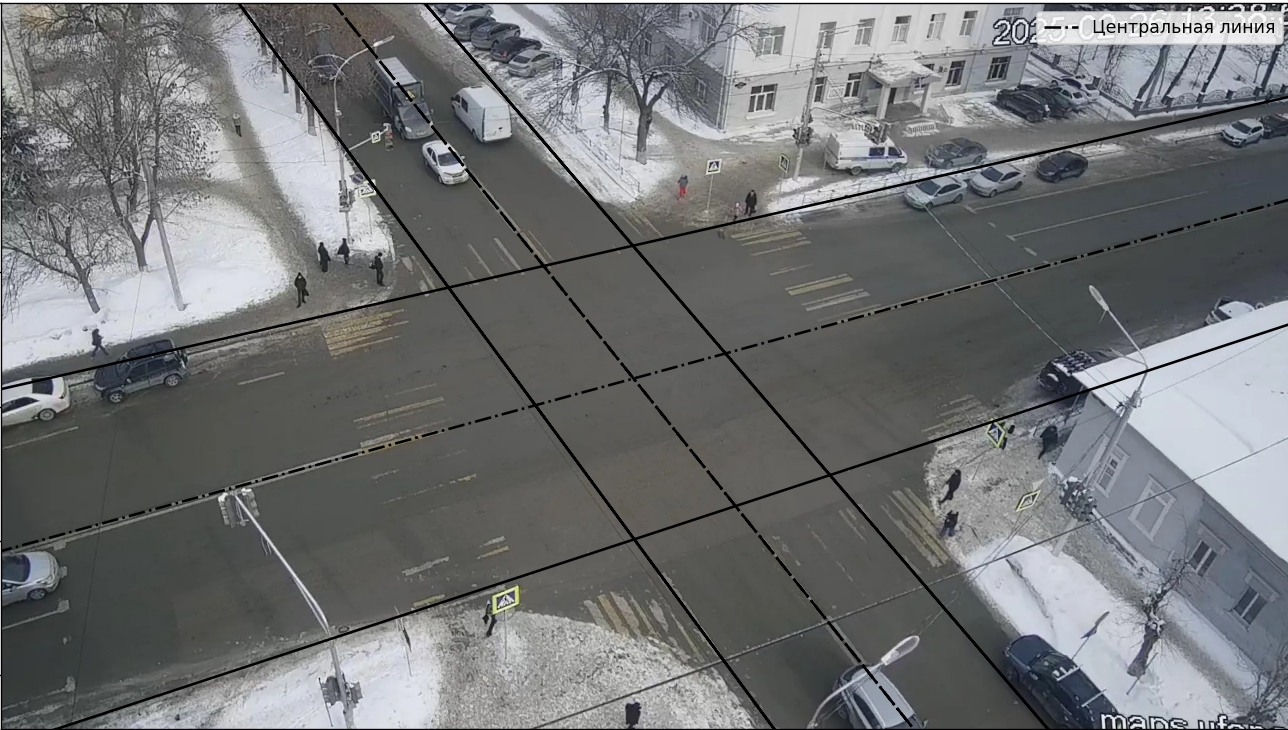
****

Рисунок 10 – Отображение линий, соответствующих ширине полосы.

Таким образом, сравнивая результаты на реальных и синтетических данных, можно сделать вывод, что метод зависит от данных, так как на его точность влияют искажения объектива (дисторсия). При этом он достаточно точен на реальных данных и позволяет находить параметры проекции с учетом этих искажений. Однако для достижения более высокой точности необходимо учитывать особенности съемки и, возможно, применять дополнительные методы компенсации искажений. В целом, метод продемонстрировал хорошую работоспособность в условиях реальных измерений и может быть использован для задач, требующих точного расчета параметров проекции на изображениях, с возможностью коррекции ошибок, связанных с искажениями объектива.

**ПРИЛОЖЕНИЕ C  
(обязательное)**

**СПИСОК ЛИТЕРАТУРЫ**

1.Jain, A. G., Saunier, N. Autocamera Calibration for Traffic Surveillance Cameras with Wide Angle Lenses [Электронный ресурс] // arXiv preprint arXiv:2001.07243. – 2020. – Режим доступа: https://doi.org/10.48550/arXiv.2001.07243.

2.Masoud, O., Papanikolopoulos, N. P. Using Geometric Primitives to Calibrate Traffic Scenes // Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). – IEEE, 2004. – DOI: 10.1109/IROS.2004.1389671.

3.Veicht, A., Sarlin, P.-E., Lindenberger, P., Pollefeys, M. GeoCalib: Learning Single-image Calibration with Geometric Optimization // European Conference on Computer Vision (ECCV), 2024. – URL: https://github.com/cvg/GeoCalib.

4.Боресков, А. В. Программирование компьютерной графики. – Москва : ДМК Пресс, 2019. – 370 с. – ISBN 978-5-97060-779-4. – Текст : электронный // Лань : электронно-библиотечная система. – URL: https://e.lanbook.com/book/131728.

5. Virtanen, P., Gommers, R., Oliphant, T. E., et al. SciPy 1.5.2: Scientific Computing with Python. – 2020. – URL: <https://www.scipy.org/>.

6.Лидердор. Горизонтальная дорожная разметка 1.14-1 [Электронный ресурс]. URL: <https://liderdor.ru/gorizontalnaya-dorozhnaya-razmetka-1-14-1>

7. OpenCV. Open Source Computer Vision Library. Version 4.x. [Электронный ресурс]. — URL: <https://opencv.org/>

**ПРИЛОЖЕНИЕ D  
(обязательное)**

**ЛИСТИНГ ПРОГРАММЫ**

**src/camera\_model.py**

import numpy as np

import cv2

from scipy.spatial.transform import Rotation

from .pointND import PointND

class Camera:

def \_\_init\_\_(self):

self.size = None

self.scene = None

self.tau = None

self.f = None

self.path = None

self.A = np.zeros((3, 3))

self.R = np.zeros((3, 3))

self.T = np.zeros((3, 1)).reshape(-1, 1)

def set\_params(self, params):

if len(params) == 5:

self.calc\_A(params[0])

self.calc\_R(params[1:4])

self.calc\_T(z=params[4])

elif len(params) == 7:

self.calc\_A(params[0])

self.calc\_R(params[1:4])

self.calc\_T(x=params[4], y=params[5], z=params[6])

def get\_scene(self):

return self.scene

def get\_f(self):

return self.f

def get\_tau(self):

return self.tau

def calc\_tau(self, height, width):

self.size = [height, width] # высота и ширина

self.tau = height / width

def load\_scene(self, path):

self.path = path

self.scene = cv2.imread(path)

height, width, channels = self.scene.shape

# print(height,width)

self.calc\_tau(height, width)

# вычисление матрицы поворота

def calc\_R(self, euler\_angles):

rot = Rotation.from\_euler('zxy', euler\_angles, degrees=True)

self.R = rot.as\_matrix()

def set\_init\_R(self, p):

self.R = np.vstack(p).transpose()

def get\_R(self, angle\_output=False, output=False):

if angle\_output:

angles = Rotation.from\_matrix(self.R).as\_euler('zxy', degrees=True)

# print(angles)

return angles

if output:

print(f'Матрица поворота:\n{self.R}')

return self.R

# вычисление столбца переноса

def calc\_T(self, x=0, y=0, z=0):

self.T = np.array([x, y, z])

def get\_T(self, output=False):

if output:

print(f'Столбец переноса:\n{self.T}')

return self.T

# вычисление внутренней матрицы

def calc\_A(self, f, using\_tau=True):

self.f = f

if using\_tau:

self.A = np.array([[f, 0, self.size[1] / 2],

[0, f \* self.tau, self.size[0] / 2],

[0, 0, 1]])

# self.A = np.array([[f, 0, 0],

# [0, f \* self.tau, 0],

# [0, 0, 1]])

else:

self.A = np.array([[f, 0, self.size[1] / 2],

[0, f \* self.size[1] / self.size[0], self.size[0] / 2],

[0, 0, 1]])

def get\_A(self, output=False):

if output:

print(f'Внутренние параметры камеры:\n{self.A}')

return self.A

# прямое преобразование

def direct\_full(self, point\_real: PointND, params=[]) -> PointND:

if len(params) == 5:

self.calc\_A(params[0])

self.calc\_R(params[1:4])

self.calc\_T(z=params[4])

elif len(params) == 7:

self.calc\_A(params[0])

self.calc\_R(params[1:4])

self.calc\_T(x=params[4], y=params[5], z=params[6])

\_T1 = -self.R @ self.T

\_RT = np.hstack([self.R, \_T1[:, np.newaxis]])

\_AT = self.A @ \_RT

\_new\_point = PointND(\_AT @ point\_real.get(out\_homogeneous=True), add\_weight=False)

return \_new\_point

def direct\_crop(self, point\_real: PointND, params=[]) -> PointND:

if len(params) == 5:

self.calc\_A(params[0])

self.calc\_R(params[1:4])

self.calc\_T(z=params[4])

elif len(params) == 7:

self.calc\_A(params[0])

self.calc\_R(params[1:4])

self.calc\_T(x=params[4], y=params[5], z=params[6])

\_T1 = -self.R @ self.T

\_RT = np.hstack([self.R, \_T1[:, np.newaxis]])

\_RT = np.delete(\_RT, 2, axis=1)

\_AT = self.A @ \_RT

\_new\_point = PointND(\_AT @ point\_real.get(out\_homogeneous=True), add\_weight=False)

return \_new\_point

def back\_crop(self, point\_image: PointND, params=[]) -> PointND:

if len(params) == 5:

self.calc\_A(params[0])

self.calc\_R(params[1:4])

self.calc\_T(z=params[4])

elif len(params) == 7:

self.calc\_A(params[0])

self.calc\_R(params[1:4])

self.calc\_T(x=params[4], y=params[5], z=params[6])

\_T1 = -self.R @ self.T

\_RT = np.hstack([self.R, \_T1[:, np.newaxis]])

\_RT = np.delete(\_RT, 2, axis=1)

\_AT = self.A @ \_RT

\_AT\_inv = np.linalg.inv(\_AT)

\_new\_point = PointND(\_AT\_inv @ point\_image.get(out\_homogeneous=True), add\_weight=False)

return \_new\_point

**src/data\_preparation.py**

from .pointND import PointND

import numpy as np

def load\_data(path):

lines = []

with open(path, 'r') as file:

for line in file:

name, cords = line.split(':')

points = eval(cords.strip())

lines.append([PointND([x, y]) for x, y in points])

return lines

def prep\_data\_angle(data):

\_data = []

if len(data) % 2 == 0:

for i in range(0, len(data), 2):

\_data.append(data[i] + data[i + 1])

return np.array(\_data)

else:

raise ValueError("Кол-во линий не четное число")

def prep\_data\_parallel(data):

\_data = []

for i in range(0, len(data) - 1):

\_data.append(data[i] + data[i + 1])

return np.array(\_data)

def load\_params(path):

with open(path, 'r') as file:

return [float(value) for value in file.readline().split()]

def prep\_data\_back\_to\_reverse(camera, data):

data = np.array(data)

data\_calc = []

for start, end in data:

start\_3d = camera.back\_crop(start)

end\_3d = camera.back\_crop(end)

data\_calc.append([camera.direct\_crop(start\_3d), camera.direct\_crop(end\_3d)])

return np.array(data\_calc)

def fun\_lines(x, start: PointND, end: PointND, orthogonal=False):

x1, y1 = start.get()

x2, y2 = end.get()

if not orthogonal:

return (x - x1) \* (y2 - y1) / (x2 - x1) + y1

else:

m = (y2 - y1) / (x2 - x1)

return (-1 / m) \* (x - x1) + y1

**src/manual\_data\_input.py**

import cv2

import numpy as np

# Глобальные переменные для хранения линий, точек и состояния рисования

lines = [] # Список для хранения линий (каждая линия - пара точек)

current\_line = [] # Текущая создаваемая линия

dragging\_point = None # (индекс\_линии, индекс\_точки) для перетаскивания

selected\_line = None # Индекс выбранной линии для редактирования

edit\_mode = False # Режим редактирования активен

# Функция для обработки кликов мышью

def click\_event(event, x, y, flags, param):

global lines, current\_line, dragging\_point, img, img\_copy, selected\_line, edit\_mode

# Если нажата левая кнопка мыши

if event == cv2.EVENT\_LBUTTONDOWN:

# Если активен режим редактирования

if edit\_mode and selected\_line is not None:

# Проверяем, не начинаем ли мы перетаскивание точки в выбранной линии

for point\_idx, point in enumerate(lines[selected\_line]):

if abs(point[0] - x) < 10 and abs(point[1] - y) < 10:

dragging\_point = (selected\_line, point\_idx)

return

else:

# Проверяем, не начинаем ли мы перетаскивание точки

for line\_idx, line in enumerate(lines):

for point\_idx, point in enumerate(line):

if abs(point[0] - x) < 10 and abs(point[1] - y) < 10:

dragging\_point = (line\_idx, point\_idx)

return

# Проверяем, не выбираем ли мы линию для редактирования

for line\_idx, line in enumerate(lines):

if len(line) == 2:

# Проверяем, находится ли клик рядом с линией

dist = point\_to\_line\_distance(line[0], line[1], (x, y))

if dist < 10: # Порог расстояния

selected\_line = line\_idx

edit\_mode = True

print(f"Выбрана линия #{line\_idx + 1} для редактирования")

redraw\_image()

return

# Если не перетаскиваем точку и не выбираем линию, то добавляем новую точку

if len(current\_line) < 2:

current\_line.append((x, y))

redraw\_image()

# Если линия завершена (две точки), добавляем её в список линий

if len(current\_line) == 2:

lines.append(current\_line.copy())

current\_line = [] # Очищаем для создания новой линии

# Если нажата правая кнопка мыши (отмена последнего действия или выход из режима редактирования)

elif event == cv2.EVENT\_RBUTTONDOWN:

if edit\_mode:

edit\_mode = False

selected\_line = None

print("Режим редактирования отключен")

elif len(current\_line) > 0:

current\_line.pop()

elif len(lines) > 0:

lines.pop()

redraw\_image()

# Если мышь перемещается с зажатой кнопкой мыши

elif event == cv2.EVENT\_MOUSEMOVE:

if dragging\_point is not None:

line\_idx, point\_idx = dragging\_point

lines[line\_idx][point\_idx] = (x, y)

redraw\_image()

# Если кнопка мыши отпущена

elif event == cv2.EVENT\_LBUTTONUP:

dragging\_point = None

# Функция для расчета расстояния от точки до линии

def point\_to\_line\_distance(line\_point1, line\_point2, point):

x1, y1 = line\_point1

x2, y2 = line\_point2

x0, y0 = point

# Вычисляем расстояние от точки до прямой

numerator = abs((y2 - y1) \* x0 - (x2 - x1) \* y0 + x2 \* y1 - y2 \* x1)

denominator = ((y2 - y1) \*\* 2 + (x2 - x1) \*\* 2) \*\* 0.5

if denominator == 0:

return ((x0 - x1) \*\* 2 + (y0 - y1) \*\* 2) \*\* 0.5 # Если точки совпадают, вернуть расстояние до точки

return numerator / denominator

# Функция для перерисовки изображения

def redraw\_image():

img[:] = img\_copy.copy()

# Рисуем все сохраненные линии

for idx, line in enumerate(lines):

# Цвет линии меняется в зависимости от индекса (для различия)

color = (255, 0, 0) # Базовый цвет - синий

if idx % 3 == 1:

color = (0, 255, 0) # Зеленый

elif idx % 3 == 2:

color = (0, 0, 255) # Красный

# Если эта линия выбрана для редактирования, выделяем её

if edit\_mode and idx == selected\_line:

line\_thickness = 3

point\_size = 7

# Добавляем текст "EDIT"

if len(line) == 2:

mid\_x = (line[0][0] + line[1][0]) // 2

mid\_y = (line[0][1] + line[1][1]) // 2 - 20

cv2.putText(img, "EDIT", (mid\_x, mid\_y), cv2.FONT\_HERSHEY\_SIMPLEX,

0.8, (0, 255, 255), 2)

else:

line\_thickness = 2

point\_size = 5

for point in line:

cv2.circle(img, point, point\_size, (0, 255, 0), -1)

if len(line) == 2:

cv2.line(img, line[0], line[1], color, line\_thickness)

# Добавляем номер линии рядом с ней

mid\_x = (line[0][0] + line[1][0]) // 2

mid\_y = (line[0][1] + line[1][1]) // 2

cv2.putText(img, str(idx + 1), (mid\_x, mid\_y), cv2.FONT\_HERSHEY\_SIMPLEX,

0.8, (255, 255, 255), 2)

# Рисуем текущую создаваемую линию

for point in current\_line:

cv2.circle(img, point, 5, (0, 255, 0), -1)

if len(current\_line) == 2:

cv2.line(img, current\_line[0], current\_line[1], (255, 0, 0), 2)

# Рисуем статус режима редактирования в верхнем левом углу

status\_text = "Режим: " + ("Редактирование" if edit\_mode else "Создание")

cv2.putText(img, status\_text, (10, 30), cv2.FONT\_HERSHEY\_SIMPLEX,

0.8, (255, 255, 255), 2)

# Показываем изображение

cv2.imshow("Calibration scene", img)

# Функция для удаления выбранной линии

def delete\_selected\_line():

global lines, selected\_line, edit\_mode

if selected\_line is not None and 0 <= selected\_line < len(lines):

del lines[selected\_line]

print(f"Линия #{selected\_line + 1} удалена")

selected\_line = None

edit\_mode = False

redraw\_image()

# Загружаем изображение

img = cv2.imread('../example/pushkin\_aksakov/image/image.webp')

if img is None:

print("Ошибка загрузки изображения. Проверьте путь к файлу.")

exit()

img\_copy = img.copy() # Создаем копию для перерисовки

# Отображаем изображение

cv2.namedWindow('Calibration scene', cv2.WINDOW\_NORMAL)

cv2.imshow("Calibration scene", img)

# Подключаем обработчик событий

cv2.setMouseCallback("Calibration scene", click\_event)

# Выводим инструкции

print("Инструкции:")

print("- Левый клик: добавить точку или начать перетаскивание существующей точки")

print("- Левый клик на линии: выбрать линию для редактирования")

print("- Правый клик: отменить последнее действие или выйти из режима редактирования")

print("- Две точки создают линию, после чего можно начать создавать следующую линию")

print("- Нажмите 'e' для переключения режима редактирования")

print("- Нажмите 'd' для удаления выбранной линии")

print("- Нажмите 's' для сохранения координат всех линий в файл")

print("- Нажмите 'q' или 'Esc' для выхода")

# Основной цикл

while True:

key = cv2.waitKey(1) & 0xFF

if key == 27 or key == ord('q'): # Esc или q для выхода

break

elif key == ord('s'): # 's' для сохранения

with open('../example/pushkin\_aksakov/marked\_data/calibration\_lines.txt', 'w') as f:

for idx, line in enumerate(lines):

f.write(f"Line {idx + 1}: {line}\n")

print(f"Сохранено {len(lines)} линий в файл 'calibration\_lines.txt'")

print(lines)

elif key == ord('e'): # 'e' для переключения режима редактирования

edit\_mode = not edit\_mode

if not edit\_mode:

selected\_line = None

print(f"Режим редактирования {'включен' if edit\_mode else 'выключен'}")

redraw\_image()

elif key == ord('d'): # 'd' для удаления выбранной линии

delete\_selected\_line()

# Закрываем окна

cv2.destroyAllWindows()

**src/new\_optimization.py**

import numpy as np

from scipy.optimize import least\_squares

from scipy.optimize import minimize

from .camera\_model import Camera

from .pointND import PointND

from .data\_preparation import fun\_lines

RESIDUALS = []

PARAMS = []

class NewOptimization:

def \_\_init\_\_(self, camera):

self.camera = camera

self.params = None

def set\_params(self, params):

self.params = params

def \_back\_project\_line\_3d(self, start2d: PointND, end2d: PointND, params):

# print(start2d.get(out\_homogeneous=True))

pre\_start3d = self.camera.back\_crop(start2d, params)

pre\_end3d = self.camera.back\_crop(end2d, params)

# print(pre\_start3d.get())

return pre\_end3d.get() - pre\_start3d.get()

def \_angle\_restrictions(self, line: np.ndarray, params):

start2d\_1, end2d\_1, start2d\_2, end2d\_2 = line

line\_1 = self.\_back\_project\_line\_3d(start2d\_1, end2d\_1, params)

line\_2 = self.\_back\_project\_line\_3d(start2d\_2, end2d\_2, params)

dot\_product = np.dot(line\_1, line\_2)

norm\_known = np.linalg.norm(line\_1)

norm\_predicted = np.linalg.norm(line\_2)

cos\_theta = np.clip(dot\_product / (norm\_known \* norm\_predicted), -1.0, 1.0)

angle\_rad = np.arccos(cos\_theta)

angle\_deg = np.degrees(angle\_rad)

return abs(angle\_deg - 90)

def \_parallel\_restrictions(self, line: np.ndarray, params):

start2d\_1, end2d\_1, start2d\_2, end2d\_2 = line

line\_1 = self.\_back\_project\_line\_3d(start2d\_1, end2d\_1, params)

line\_2 = self.\_back\_project\_line\_3d(start2d\_2, end2d\_2, params)

dot\_product = np.dot(line\_1, line\_2)

norm\_known = np.linalg.norm(line\_1)

norm\_predicted = np.linalg.norm(line\_2)

cos\_theta = np.clip(dot\_product / (norm\_known \* norm\_predicted), -1.0, 1.0)

return abs(1 - cos\_theta)

def \_point\_to\_point(self, line: np.ndarray, params, log\_calc=False):

start, end = line

line = self.\_back\_project\_line\_3d(start, end, params)

dist\_calc = np.linalg.norm(line)

dist = 4.2

return abs(dist\_calc - dist)

def \_dist\_between\_line(self, line: np.ndarray, y\_dist, params):

start2d\_1, end2d\_1, start2d\_2, end2d\_2 = line

x = np.linspace(-10, 10, 100)

y\_predict = fun\_lines(x, self.camera.back\_crop(start2d\_2, params),

self.camera.back\_crop(end2d\_2, params)) + y\_dist

y\_known = fun\_lines(x, self.camera.back\_crop(start2d\_1, params), self.camera.back\_crop(end2d\_1, params))

# print(np.array(y\_known) - np.array(y\_predict))

# print(np.linalg.norm(np.array(y\_known) - np.array(y\_predict)))

return np.linalg.norm(np.array(y\_known) - np.array(y\_predict))

def target\_function(self, params, data):

params[1] = self.periodic\_bound(params[1], -360, 360)

params[2] = self.periodic\_bound(params[2], -360, 360)

params[3] = self.periodic\_bound(params[3], -360, 360)

residuals = []

# data\_angle = data['angle'] if 'angle' in data and data['angle'].size > 0 else []

data\_parallel\_line\_1 = data['parallel-1'] if 'parallel-1' in data and data['parallel-1'].size > 0 else []

data\_point\_to\_point = data['point\_to\_point'] if 'point\_to\_point' in data and data[

'point\_to\_point'].size > 0 else []

data\_parallel\_line\_2 = data['parallel-2'] if 'parallel-2' in data and data['parallel-2'].size > 0 else []

# for \_data in data\_angle:

# # # print(f'Angle: {self.\_angle\_restrictions(\_data, params)}')

# residuals.append(self.\_angle\_restrictions(\_data, params))

for dist, \_data in zip([-11, 11], data\_parallel\_line\_1):

residuals.append(self.\_parallel\_restrictions(\_data, params))

# residuals.append(self.\_dist\_between\_line(\_data, dist, params))

for dist, \_data in zip([-7, 7], data\_parallel\_line\_2):

residuals.append(self.\_parallel\_restrictions(\_data, params))

# residuals.append(self.\_dist\_between\_line(\_data, dist, params))

for \_data in data\_point\_to\_point:

residuals.append(self.\_point\_to\_point(\_data, params))

RESIDUALS.append(np.array(residuals))

PARAMS.append(params)

return np.concatenate([np.ravel(res) for res in residuals])

def periodic\_bound(self, value, lower\_bound, upper\_bound):

range\_width = upper\_bound - lower\_bound

return lower\_bound + (value - lower\_bound) % range\_width

def back\_projection(self, data):

self.params = [1400, -180, 0.91236625, -180.6947188, 15]

bounds = ([900, -360, -360, -360, 5], [4000, 360, 360, 360, 60])

# self.params = np.random.uniform(low=bounds[0], high=bounds[1])

result = least\_squares(self.target\_function, self.params, args=(data,), method='dogbox',

verbose=2,

bounds=bounds,

# loss='soft\_l1',

jac='3-point'

# ftol=1e-8, xtol=1e-8, gtol=1e-8

)

# result = minimize(self.target\_function, self.params, args=(data,), method='Nelder-Mead',

# bounds=list(zip(bounds[0], bounds[1])),options={'maxiter': 1000, 'disp': True})

print(\*np.around(result.x, 2))

**src/plot.py**

import cv2

import matplotlib.pyplot as plt

from enum import Enum, auto

from pathlib import Path

import numpy as np

from .camera\_model import Camera

from .pointND import PointND

class DisplayMode(Enum):

INTERACTIVE = auto()

JUPYTER = auto()

SAVE = auto()

class ProjectionMode(Enum):

DIRECT = auto()

BACK = auto()

class Plot:

def \_\_init\_\_(self, camera: Camera):

self.camera = camera

self.scene\_plot = self.camera.get\_scene().copy() # копия сцены

self.overlay = self.scene\_plot.copy() # слой сцены

self.mode = DisplayMode.INTERACTIVE

def set\_mode(self, mode: DisplayMode):

self.mode = mode

def \_transform\_pointND\_to\_cv2\_format(self, point: PointND):

if point:

return tuple(map(int, point.get()))

def \_draw\_point\_with\_label(self, point2d, coords):

cv2.circle(self.overlay, point2d, 5, (0, 0, 255), -1)

if len(coords) == 2:

text = f"({coords[0]:.1f}, {coords[1]:.1f})"

else:

text = f"({coords[0]:.1f}, {coords[1]:.1f}, {coords[2]:.1f} )"

cv2.putText(self.overlay, text, (point2d[0] + 5, point2d[1] - 5),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 1, cv2.LINE\_AA)

def draw\_point(self, points: np.ndarray, params=None, mode=ProjectionMode.DIRECT):

if self.mode == DisplayMode.JUPYTER:

self.overlay = self.scene\_plot.copy()

params = params or []

for point in points:

point\_plot = None

if params:

if ProjectionMode.DIRECT:

point\_plot = self.camera.direct\_full(point, params)

elif ProjectionMode.BACK:

point\_plot = self.camera.back\_crop(point, params)

else:

point\_plot = point

point\_plot = self.\_transform\_pointND\_to\_cv2\_format(point\_plot)

self.\_draw\_point\_with\_label(point\_plot, point.get())

alpha = 0.8

cv2.addWeighted(self.overlay, alpha, self.scene\_plot, 1 - alpha, 0, self.scene\_plot)

def draw\_line(self, lines: np.ndarray, params=None, mode=ProjectionMode.DIRECT, color=(255, 0, 0), thickness=2):

if self.mode == DisplayMode.JUPYTER:

self.overlay = self.scene\_plot.copy()

params = params or []

for line in lines:

start, end = line

start\_plot, end\_plot = None, None

if params:

if mode == ProjectionMode.DIRECT:

start\_plot = self.camera.direct\_full(start, params)

end\_plot = self.camera.direct\_full(end, params)

elif mode == ProjectionMode.BACK:

start\_plot = self.camera.back\_crop(start, params)

end\_plot = self.camera.back\_crop(end, params)

else:

start\_plot = start

end\_plot = end

start\_plot = self.\_transform\_pointND\_to\_cv2\_format(start\_plot)

end\_plot = self.\_transform\_pointND\_to\_cv2\_format(end\_plot)

self.\_draw\_point\_with\_label(start\_plot, start.get())

self.\_draw\_point\_with\_label(end\_plot, end.get())

cv2.line(

self.overlay,

start\_plot,

end\_plot,

color,

thickness

)

alpha = 0.8

cv2.addWeighted(self.overlay, alpha, self.scene\_plot, 1 - alpha, 0, self.scene\_plot)

def visible(self, mode: DisplayMode = None):

mode = mode or self.mode

if mode == DisplayMode.SAVE:

filename = Path(self.camera.path).name

print('calib\_' + filename)

cv2.imwrite('calibline\_' + filename, self.overlay)

elif mode == DisplayMode.JUPYTER:

scene\_rgb = cv2.cvtColor(self.overlay, cv2.COLOR\_BGR2RGB)

plt.figure(figsize=(10, 8))

# plt.title()

plt.imshow(scene\_rgb)

plt.axis('off')

plt.show()

elif mode == DisplayMode.INTERACTIVE:

cv2.namedWindow('Calibration scene', cv2.WINDOW\_NORMAL)

initial\_width = 1000 # Ширина окна

initial\_height = 700 # Высота окна

cv2.resizeWindow('Calibration scene', initial\_width, initial\_height)

cv2.imshow('Calibration scene', self.overlay)

cv2.waitKey(0)

cv2.destroyAllWindows()

**src/pointND.py**

import numpy as np

class PointND:

def \_\_init\_\_(self, coord, add\_weight=True):

coord = np.asarray(coord)

if len(coord) + 1 in [3, 4] and add\_weight:

coord = np.append(coord, 1)

self.coord = coord

def set(self, coord):

self.coord = np.append(coord, 1) if len(coord) + 1 == len(self.coord) else coord

def get(self, out\_homogeneous=False):

return self.coord if out\_homogeneous else self.coord[:-1] / self.coord[-1]

def get\_type(self):

dim = len(self.coord) - 1

return f"{dim}D"

def set\_Z(self, z):

if len(self.coord) > 3:

self.coord[2] = z

else:

raise ValueError("Объект не является 3D точкой")

**draw.py**

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

import numpy as np

from scipy.spatial.transform import Rotation

from src.camera\_model import Camera

from src.pointND import PointND

from src.new\_optimization import NewOptimization,RESIDUALS

from src.data\_preparation import fun\_lines, load\_params, load\_data

def init(h):

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

ax.zaxis.line.set\_color((1.0, 1.0, 1.0, 0.0)) # Ось Z

ax.xaxis.line.set\_color((1.0, 1.0, 1.0, 0.0)) # Ось X

ax.yaxis.line.set\_color((1.0, 1.0, 1.0, 0.0)) # Ось Y

ax.xaxis.set\_tick\_params(labelleft=False, labelbottom=False) # Убираем метки для оси X

ax.yaxis.set\_tick\_params(labelleft=False, labelbottom=False) # Убираем метки для оси Y

ax.zaxis.set\_tick\_params(labelleft=False, labelbottom=False) # Убираем метки для оси Z

ax.xaxis.set\_ticks\_position('none') # Убираем засечки для оси X

ax.yaxis.set\_ticks\_position('none') # Убираем засечки для оси Y

ax.zaxis.set\_ticks\_position('none') # Убираем засечки для оси Z

# Настройка углов обзора

ax.view\_init(elev=20, azim=30)

ax.set\_proj\_type('persp')

ax.set\_zlim(0, h + 10)

ax.set\_xlim(-40, 40)

ax.set\_ylim(-40, 40)

return ax

def plot\_axies(position, angles=[]):

if not angles:

ax.quiver(\*position, 15, 0, 0, color='black')

ax.quiver(\*position, 0, 15, 0, color='black')

ax.quiver(\*position, 0, 0, 15, color='black')

ax.scatter(0, 0, 0, marker='^', s=100, color='red', label='Мировая система координат')

text\_size = 12

ax.text(position[0] + 15, position[1] + 1, position[2], 'X', color='black', fontsize=text\_size)

ax.text(position[0], position[1] + 15, position[2], 'Y', color='black', fontsize=text\_size)

ax.text(position[0], position[1], position[2] + 15, 'Z', color='black', fontsize=text\_size)

else:

rot = Rotation.from\_euler('zxy', angles, degrees=True).as\_matrix()

transform = np.eye(4)

transform[:3, :3] = rot

transform[:3, 3] = -rot @ position

x\_position = transform @ np.array([15, 0, 0, 1])

y\_position = transform @ np.array([0, 15, 0, 1])

z\_position = transform @ np.array([0, 0, 15, 1])

origin = transform @ np.array([0, 0, 0, 1])

# print(f'Положение камеры:\nx: {x\_position[:-1]}\ny: {y\_position[:-1]}\nz: {z\_position[:-1]}')

distances = np.linalg.norm(transform[:3, 3])

ax.scatter(\*transform[:3, 3], color='red', label='Система координат камеры камеры')

# label=f'{np.around(transform[:3, 3], 2)}, расстонияние до центра {round(distances, 2)}')

ax.quiver(\*origin[:-1], \*(x\_position[:-1] - origin[:-1]), color='black')

ax.quiver(\*origin[:-1], \*(y\_position[:-1] - origin[:-1]), color='black')

ax.quiver(\*origin[:-1], \*(z\_position[:-1] - origin[:-1]), color='black')

text\_size = 12

ax.text(\*x\_position[:-1], 'X', color='black', fontsize=text\_size)

ax.text(\*y\_position[:-1], 'Y', color='black', fontsize=text\_size)

ax.text(\*z\_position[:-1], 'Z', color='black', fontsize=text\_size)

ax.legend(loc='upper center')

def get\_normals(x, y, length=4):

"""

Строит нормальные (перпендикулярные) отрезки длиной `length` для заданных точек линии `x, y`.

Возвращает:

- norm\_lines\_x: список массивов x-координат отрезков

- norm\_lines\_y: список массивов y-координат отрезков

"""

norm\_lines\_x = []

norm\_lines\_y = []

# Выбираем точки для построения нормалей

indices = [73, 69] # Можно выбрать другие индексы

for idx in indices:

if idx >= len(x) - 1: # Проверяем границы массива

continue

# Определяем направление линии в данной точке

dx = x[idx + 1] - x[idx]

dy = y[idx + 1] - y[idx]

# Вычисляем нормальный вектор

norm = np.array([-dy, dx])

norm = length \* norm / np.linalg.norm(norm) # Делаем его длиной `length`

# Вычисляем координаты концов нормального отрезка

norm\_x = [x[idx] - norm[0] / 2, x[idx] + norm[0] / 2]

norm\_y = [y[idx] - norm[1] / 2, y[idx] + norm[1] / 2]

norm\_lines\_x.append(norm\_x)

norm\_lines\_y.append(norm\_y)

return norm\_lines\_x, norm\_lines\_y

# эталонные значения

camera = Camera()

camera.load\_scene('../pushkin\_aksakov/image/image.webp')

params = load\_params('../pushkin\_aksakov/marked\_data/calib\_data.txt')

camera.set\_params(params)

ax = init(params[-1])

plot\_axies([0, 0, 0])

plot\_axies([0, 0, params[-1]], params[1:4])

data = {}

data['parallel-1'] = []

data['parallel-2'] = []

data['angle'] = []

data['point\_to\_point'] = []

# Паралельные линии

start, end = load\_data('../pushkin\_aksakov/marked\_data/parallel\_lines\_1.txt')[1]

start3d = camera.back\_crop(start)

end3d = camera.back\_crop(end)

print(np.linalg.norm(end3d.get() - start3d.get()))

for y\_dist in [-10, 0, 10]:

x = np.linspace(-60, 25, 100)

y = fun\_lines(x, start3d, end3d) - y\_dist

points = [camera.direct\_crop(PointND([xi, yi])) for xi, yi in zip(x, y)]

x\_new, y\_new = zip(\*[p.get() for p in points])

data['parallel-1'].append(np.array([x\_new, y\_new]))

plt.plot(x, y, color='black')

norm\_x, norm\_y = get\_normals(x, y, length=4)

for i in range(len(norm\_x)):

plt.plot(norm\_x[i], norm\_y[i], color='black', label=f"Normal {i + 1}")

points = [camera.direct\_crop(PointND([xi, yi])) for xi, yi in zip(norm\_x[i], norm\_y[i])]

x\_new, y\_new = zip(\*[p.get() for p in points])

data['point\_to\_point'].append(np.array([x\_new, y\_new]))

# Ортогональные линии

x = np.linspace(5, 13, 100)

y = fun\_lines(x, start3d, end3d, orthogonal=True)

plt.plot(x, y, color='black')

points = [camera.direct\_crop(PointND([xi, yi])) for xi, yi in zip(x, y)]

x\_new, y\_new = zip(\*[p.get() for p in points])

data['angle'].append(np.array([x\_new, y\_new]))

start, end = load\_data('../pushkin\_aksakov/marked\_data/parallel\_lines\_2.txt')[1]

start3d = camera.back\_crop(start)

end3d = camera.back\_crop(end)

print(np.linalg.norm(end3d.get() - start3d.get()))

for y\_dist in [-10, 0, 10]:

x = np.linspace(-17, 17, 100)

y = fun\_lines(x, start3d, end3d) - y\_dist

points = [camera.direct\_crop(PointND([xi, yi])) for xi, yi in zip(x, y)]

x\_new, y\_new = zip(\*[p.get() for p in points])

data['parallel-2'].append(np.array([x\_new, y\_new]))

plt.plot(x, y, color='black')

# Ортогональные линии

x = np.linspace(0, 4, 100)

y = fun\_lines(x, start3d, end3d, orthogonal=True)

plt.plot(x, y, color='black')

points = [camera.direct\_crop(PointND([xi, yi])) for xi, yi in zip(x, y)]

x\_new, y\_new = zip(\*[p.get() for p in points])

data['angle'].append(np.array([x\_new, y\_new]))

plt.show()

data\_optimize = {}

data\_optimize['parallel-1'] = []

data\_optimize['parallel-2'] = []

data\_optimize['angle'] = []

data\_optimize['point\_to\_point'] = []

for x, y in data['parallel-1']:

plt.plot(x, y, color='black')

data\_optimize['parallel-1'].append([PointND([x[0], y[0]]), PointND([x[-1], y[-1]])])

for x, y in data['parallel-2']:

plt.plot(x, y, color='black')

data\_optimize['parallel-2'].append([PointND([x[0], y[0]]), PointND([x[-1], y[-1]])])

for x, y in data['angle']:

plt.plot(x, y, color='black')

data\_optimize['angle'].append([PointND([x[0], y[0]]), PointND([x[-1], y[-1]])])

for x, y in data['point\_to\_point']:

plt.plot(x, y, color='black')

data\_optimize['point\_to\_point'].append([PointND([x[0], y[0]]), PointND([x[-1], y[-1]])])

plt.xlim(0, 1920)

plt.ylim(0, 1080)

plt.gca().invert\_yaxis()

plt.show()

curr = []

for i in range(1, len(data\_optimize['parallel-1'])):

curr.append(data\_optimize['parallel-1'][i - 1] + data\_optimize['parallel-1'][i])

data\_optimize['parallel-1'] = np.array(curr)

curr = []

for i in range(1, len(data\_optimize['parallel-2'])):

curr.append(data\_optimize['parallel-2'][i - 1] + data\_optimize['parallel-2'][i])

data\_optimize['parallel-2'] = np.array(curr)

data\_optimize['angle'] = np.array

data\_optimize['point\_to\_point'] = np.array(data\_optimize['point\_to\_point'])

optimize = NewOptimization(camera)

optimize.back\_projection(data\_optimize)

HIST = [np.sum(values) for values in RESIDUALS]

plt.title('График погрешности')

plt.ylabel('Точность')

plt.xlabel('Количество итераций')

plt.plot(np.arange(0, len(HIST)), HIST)

plt.show()

**example\_back.py**

from src.camera\_model import Camera

from src.new\_optimization import NewOptimization, RESIDUALS, PARAMS

from src.initsolution import calc\_init\_camera

from src.plot import Plot, DisplayMode, ProjectionMode

from src.pointND import PointND

from src.distance import gps\_to\_enu

from src.data\_preparation import load\_data, prep\_data\_parallel, prep\_data\_angle, load\_params, prep\_data\_back\_to\_reverse, \

fun\_lines

import numpy as np

import matplotlib.pyplot as plt

import matplotlib

matplotlib.use("TkAgg")

camera = Camera()

camera.load\_scene('image/image.webp')

# # Отрисовка исходных линий

# plot = Plot(camera)

# # plot.draw\_line(load\_data('marked\_data/angle\_lines.txt'))

# plot.draw\_line(load\_data('marked\_data/parallel\_lines\_1.txt'))

# plot.draw\_line(load\_data('marked\_data/parallel\_lines\_2.txt'))

# plot.draw\_line(load\_data('marked\_data/point\_to\_point.txt'))

# plot.visible()

# # # #

# # # Оптимизация

data = {

# 'angle': prep\_data\_angle(load\_data('marked\_data/angle\_lines.txt')),

'parallel-1': prep\_data\_parallel(load\_data('marked\_data/parallel\_lines\_1.txt')),

'point\_to\_point': np.array(load\_data('marked\_data/point\_to\_point.txt')),

'parallel-2': prep\_data\_parallel(load\_data('marked\_data/parallel\_lines\_2.txt')),

}

print(data)

# optimize = NewOptimization(camera)

# optimize.back\_projection(data)

# # #

# # # # # Отрисовка результатов оптимизации

# HIST = [np.sum(values) for values in RESIDUALS]

#

# plt.figure(1)

# plt.subplot(1, 2, 1)

# plt.title('График погрешности')

# plt.ylabel('Точность')

# plt.xlabel('Количество итераций')

# plt.plot(np.arange(0, len(HIST)), HIST)

#

# plt.subplot(1, 2, 2)

# plt.plot(RESIDUALS[0], label='Первая итерация')

# plt.plot(RESIDUALS[-1], label='Последняя итерация')

# plt.axvspan(0, 3, color='lightgrey', alpha=0.5,label='Параллельные прямые')

# # plt.axvspan(1, 3, color='lightgrey', alpha=0.5)

# plt.axvspan(3, 14, color='darkgrey', alpha=0.5,label='Расстояние от точки до точки')

# # plt.axvline(x=1, color='black', linestyle='--') # Вертикальная линия на X=5

# # plt.axvline(x=3, color='black', linestyle='--')

# plt.text(2.5, 12, 'Область 1', horizontalalignment='center', verticalalignment='center', color='black', fontsize=10)

# plt.text(7.5, 12, 'Область 2', horizontalalignment='center', verticalalignment='center', color='black', fontsize=10)

# plt.title('Погрешность для всех наборов данных')

# plt.ylabel('Погрешность')

# plt.xlabel('Наборы данных')

# plt.legend()

# plt.show()

# PARAMS = np.array(PARAMS)

#

# plt.plot(PARAMS[:, 0], label='Фокусное расстояние')

# plt.plot(PARAMS[:, 1], label='Вращение вокруг Z')

# plt.plot(PARAMS[:, 2], label='Вращение вокруг X')

# plt.plot(PARAMS[:, 3], label='Вращение вокруг Y')

# plt.plot(PARAMS[:, 4], label='Высота')

# plt.legend()

# plt.show()

# Тесты

# camera.set\_params(load\_params('marked\_data/calib\_data.txt'))

# optimize = NewOptimization(camera)

#

# data\_calc = prep\_data\_back\_to\_reverse(camera,

# load\_data('marked\_data/point\_to\_point.txt') + load\_data(

# 'marked\_data/parallel\_lines\_1.txt') + load\_data(

# 'marked\_data/parallel\_lines\_2.txt'))

# plot = Plot(camera)

# plot.draw\_line(data\_calc, thickness=7)

# plot.draw\_line(load\_data('marked\_data/parallel\_lines\_1.txt'), color=(0, 255, 0))

# plot.draw\_line(load\_data('marked\_data/parallel\_lines\_2.txt'), color=(0, 255, 0))

# plot.draw\_line(load\_data('marked\_data/point\_to\_point.txt'), color=(0, 255, 0))

# plot.visible(DisplayMode.INTERACTIVE)

# data = load\_data('calibration\_lines.txt')

# print(np.linalg.norm(optimize.\_back\_project\_line\_3d(\*data[0], load\_params('calib\_data.txt'))))

# Прямая линия

# camera = Camera()

# camera.load\_scene('image/crossroads\_not\_dist\_ver2.webp')

# camera.set\_params(load\_params('marked\_data/calib\_data.txt'))

#

# plot\_coord = []

# for start, end in load\_data('marked\_data/parallel\_lines\_1.txt'):

# start3d = camera.back\_crop(start)

# end3d = camera.back\_crop(end)

# plot\_coord.append([start3d, end3d])

#

# for i in range(1, len(plot\_coord)):

# start1, end1 = plot\_coord[i - 1]

# start2, end2 = plot\_coord[i]

# plt.plot([start1.get()[0], end1.get()[0]], [start1.get()[1], end1.get()[1]],

# label=f'Расстояние\nНачало: {np.linalg.norm(start2.get() - start1.get())}\nКонец: {np.linalg.norm(end2.get() - end1.get())}')

# plt.scatter(start1.get()[0], start1.get()[1])

# start2, end2 = plot\_coord[-1]

# plt.plot([start2.get()[0], end2.get()[0]], [start2.get()[1], end2.get()[1]])

# Прямая линия продолжение

# camera = Camera()

# camera.load\_scene('image/image.webp')

# camera.set\_params(load\_params('marked\_data/calib\_data.txt'))

#

# import cv2

#

# scene = cv2.imread('image/crossroads\_not\_dist.jpg')

# scene\_rgb = cv2.cvtColor(scene, cv2.COLOR\_BGR2RGB)

# plt.imshow(scene\_rgb)

# coord1 = []

# coord2 = []

# for i, (start, end) in enumerate(load\_data('marked\_data/parallel\_lines\_2.txt')):

# start3d = camera.back\_crop(start)

# end3d = camera.back\_crop(end)

# x = np.linspace(-100, 100, 100)

# y = fun\_lines(x, start3d, end3d)

# points = [camera.direct\_crop(PointND([xi, yi])) for xi, yi in zip(x, y)]

# x\_new, y\_new = zip(\*[p.get() for p in points])

# plt.scatter([start.get()[0], end.get()[0]], [start.get()[1], end.get()[1]])

# plt.plot(x\_new, y\_new, label=f'Transformed Line 1 - {i}')

# coord1.append(np.array([x, y]))

# На известных данных

# for y\_dist in [-12, 0, 12]:

# start, end = load\_data('marked\_data/parallel\_lines\_2.txt')[1]

# start3d = camera.back\_crop(start)

# end3d = camera.back\_crop(end)

# x = np.linspace(-100, 100, 100)

# y = fun\_lines(x, start3d, end3d) - y\_dist

# points = [camera.direct\_crop(PointND([xi, yi])) for xi, yi in zip(x, y)]

# x\_new, y\_new = zip(\*[p.get() for p in points])

# if y\_dist == 0:

# plt.plot(x\_new, y\_new, label='Центральная линия', color='black', ls='-.')

# else:

# plt.plot(x\_new, y\_new, color='black')

#

# for y\_dist in [-11.5, 0, 11.5]:

# start, end = load\_data('marked\_data/parallel\_lines\_1.txt')[1]

# start3d = camera.back\_crop(start)

# end3d = camera.back\_crop(end)

# x = np.linspace(-100, 100, 100)

# y = fun\_lines(x, start3d, end3d) - y\_dist

# points = [camera.direct\_crop(PointND([xi, yi])) for xi, yi in zip(x, y)]

# x\_new, y\_new = zip(\*[p.get() for p in points])

# if y\_dist == 0:

# plt.plot(x\_new, y\_new, color='black', ls='-.')

# else:

# plt.plot(x\_new, y\_new, color='black')

# #

# plt.xlim(0, 1920)

# plt.ylim(0, 1080)

# plt.gca().invert\_yaxis()

# plt.legend()

# plt.show()

#

# # Создание синтетических данных

# camera = Camera()

# camera.load\_scene('image/image.webp')

# camera.set\_params(load\_params('marked\_data/calib\_data.txt'))

#

# start, end = load\_data('marked\_data/parallel\_lines\_1.txt')[1]

# start3d = camera.back\_crop(start)

# end3d = camera.back\_crop(end)

#

# print(np.linalg.norm(end3d.get() - start3d.get()))

# y\_dist = 0

# x = np.linspace(-25, 25, 100)

# y = fun\_lines(x, start3d, end3d) - y\_dist

# points = [camera.direct\_crop(PointND([xi, yi])) for xi, yi in zip(x, y)]

# x\_new, y\_new = zip(\*[p.get() for p in points])

# plt.plot([start3d.get()[0], end3d.get()[0]], [start3d.get()[1], end3d.get()[1]])

# plt.plot(x, y)

# plt.show()

1. При проведении практики в профильной организации руководителем практики от кафедры и руководителем практики от профильной организации составляется совместный рабочий график (план) проведения практики. [↑](#footnote-ref-1)