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

Уфа – 2025 г.

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] _____/ _____
подпись И.О. Фамилия

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

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.

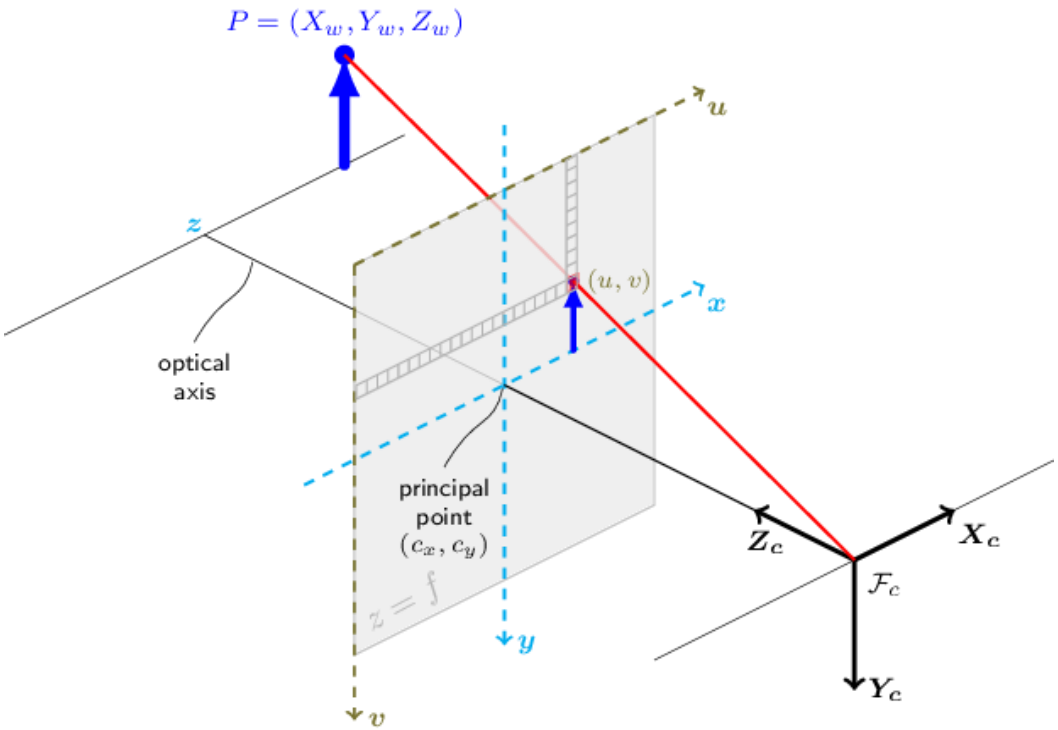

Рисунок 1 – Модель камеры-обскуры

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

$$s\,p = A\,[R|t]P_w, \tag{1}$$

где $P_w \in R^4$ – трехмерная точка в мировой системе координат (используются однородные координаты),

$p = [u, v, 1]^T \in R^3$ – двумерный пиксель в плоскости изображения (используются однородные координаты),

A – внутренняя матрица камеры,

R и t – 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 A проецирует 3D-точки, заданные в системе координат камеры, в 2D-пиксельные координаты то есть (2):

$$p = AP_c s \tag{2}$$

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

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

Матрица внутренних параметров A 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 $[R|t]$ 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 $x' = X_c/Z_c$ и $y' = Y_c/Z_c$:

$$Z_c \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}. \tag{4}$$

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

$$P_c = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} P_w, \tag{5}$$

то есть матрица однородного преобразования состоит из $R \in \mathbb{R}^{3\times3}$ – матрицы вращения, и $t \in \mathbb{R}^{3\times1}$ – вектора переноса:

$$\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{6}$$

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

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{9}$$

Для более компактного представления матрица поворота $R \in \mathbb{R}^{3\times3}$ 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 X мировой системы координат на угол α:

$$M_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \tag{10}$$

Для вращения вокруг оси Y мировой системы координат на угол β:

$$M_y(\beta) = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \tag{11}$$

Для вращения вокруг оси Z мировой системы координат на угол γ:

$$M_z(\gamma) = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (12)$$

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

$$R = M_z(\gamma) M_x(\beta) M_y(\alpha) \quad (13)$$

Раскрывая (13), получим:

$$R = \begin{bmatrix} \cos(\gamma)\cos(\beta) - \sin(\gamma)\sin(\alpha)\sin(\beta) & -\sin(\gamma)\cos(\alpha) & \cos(\gamma)\sin(\beta) + \sin(\gamma)\sin(\alpha)\cos(\beta) \\ \sin(\gamma)\cos(\beta) + \cos(\gamma)\sin(\alpha)\sin(\beta) & \cos(\gamma)\cos(\alpha) & \sin(\gamma)\sin(\beta) - \cos(\gamma)\sin(\alpha)\cos(\beta) \\ -\cos(\alpha)\sin(\beta) & \sin(\alpha) & \cos(\alpha)\cos(\beta) \end{bmatrix}$$

Рассмотренная выше модель камеры обскуры (9) задает проекцию $3D$ – точки в $2D$ 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 ($Z = 0$). 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:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & \frac{W}{2} \\ 0 & f\tau & \frac{H}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & 0 \\ r_{21} & r_{22} & 0 \\ r_{31} & r_{32} & -t_z r_{33} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix}, \quad (14)$$

где W, H – количество пикселей на изображение по горизонтали и вертикали соответственно,

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

$[u, v, 1]^T \in R^3$ – двумерный пиксель в плоскости изображения (используются однородные координаты),

$[X_w, Y_w, 1]^T \in R^3$ – точка в мировой системе координат в плоскости дороги $Z = 0$ (используются однородные координаты),

t_z – высота над плоскостью дороги $Z = 0$.

Отношения количества пикселей выражается:

$$\tau = \frac{H}{W}, \quad (15)$$

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

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

$$\begin{bmatrix} f & 0 & \frac{W}{2} \\ 0 & f\tau & \frac{H}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & 0 \\ r_{21} & r_{22} & 0 \\ r_{31} & r_{32} & -t_z r_{33} \end{bmatrix} = ART \quad (16)$$

и получим:

$$s \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix} = ART^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad (17)$$

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.

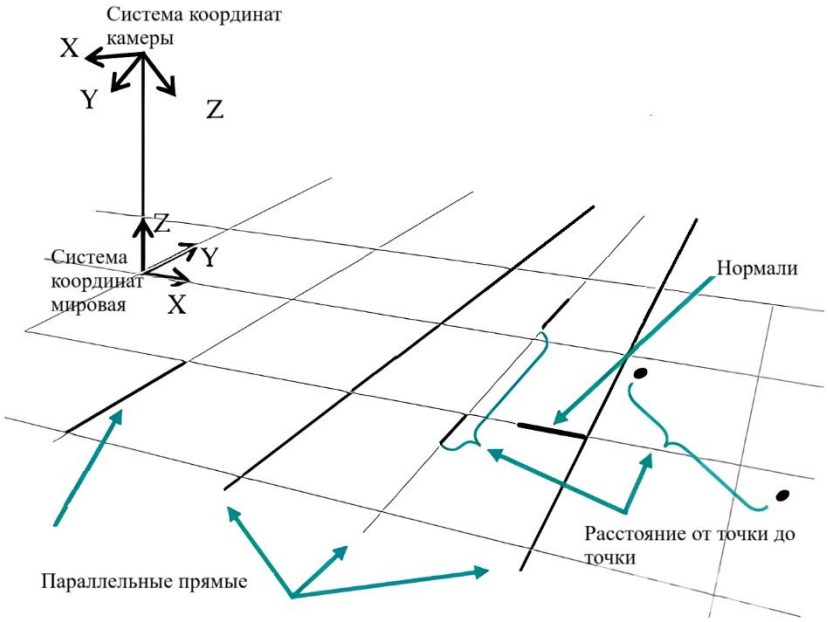

Рисунок 2 – Геометрические ограничения на сцене.

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 A_i, где $i = \overline{2,N}$ такое что $A_i = (x_i', x_i'')$, где $x_i', x_i'' \in R^3$ начало и конец отрезка на изображение. Спроецируем в мировые координаты точки $x_i', x_i'' \in R^3$ по (17) и получим:

$$X_i' = ART^{-1}x_i', X_i'' = ART^{-1}x_i'' \tag{18}$$

Вычитая X_i' из X_i'', получим вектор:

$$\overrightarrow{L_\iota} = X_i' - X_i'', \tag{19}$$

Вычисляя аналогично для прямой A_{i-1} получим соответственно для двух векторов $\overrightarrow{L_\iota}, \overrightarrow{L_{\iota-1}}$ следующую функцию ошибки для параллельных прямых:

$$f_1\left(\overrightarrow{L_\iota}, \overrightarrow{L_{\iota-1}}\right) = |1 - \frac{\overrightarrow{L_\iota}\,\overrightarrow{L_{\iota-1}}}{\left|\left|\overrightarrow{L_\iota}\right|\right|\left|\left|\overrightarrow{L_{\iota-1}}\right|\right|}| \tag{20}$$

2. 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 A_i, B_i, где $i = \overline{1, N}$ такое что $A_i = (x_i', x_i'')$ и $B_i = (x_i''', x_i'''')$, где $x_i', x_i'', x_i''', x_i'''' \in R^3$ начало и конец отрезка на изображение. Аналогично (18) и (19) спроецируем точки $x_i', x_i'', x_i''', x_i'''' \in R^3$ в мировые координаты и получим 2 вектора $\overrightarrow{L_i^A}, \overrightarrow{L_i^B}$. Следовательно функция ошибки для данного вида ограничения будет выглядеть:

$$f_2\left(\overrightarrow{L_i^A}, \overrightarrow{L_i^B}\right) = |90° - \arccos\left(\frac{\overrightarrow{L_i^A}\,\overrightarrow{L_i^B}}{\left\|\overrightarrow{L_i^A}\right\|\left\|\overrightarrow{L_i^B}\right\|}\right)| \tag{21}$$

3. 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 A_i, d_i, где $i = \overline{1, N}$ такое что $A_i = (x_i', x_i'')$, $d_i \in R$, где $x_i', x_i'' \in R^3$ начало и конец отрезка на изображение. Аналогично (18) и (19) спроецируем точки $x_i', x_i'' \in R^3$ в мировые координаты и получим вектор L_i. Из этого следует, что функция ошибки для данного вида ограничения будет следующая:

$$f_3\left(\overrightarrow{L_\iota}, d_i\right) = \left| d_i - \left\|\overrightarrow{L_\iota}\right\| \right| \tag{22}$$

Тогда совмещая все ограничения сцены, получим следующий целевой функционал:

$$F(Y) = \sum_{i=1}^{N} C_1 f_1\left(\overrightarrow{L_\iota}, \overrightarrow{L_{\iota-1}}\right) + \sum_{i=1}^{M} C_2 f_2\left(\overrightarrow{L_i^A}, \overrightarrow{L_i^B}\right) + \sum_{i=1}^{K} C_3 f_3\left(\overrightarrow{L_\iota}, d_i\right), \tag{23},$$

где $C_1, C_2 -$ весовые коэффициенты, $Y = [f, \alpha, \beta, \gamma, h]$ – 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 В
(обязательное)

РЕЗУЛЬТАТЫ РАБОТЫ

Приведя пример расчета на синтетических 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.

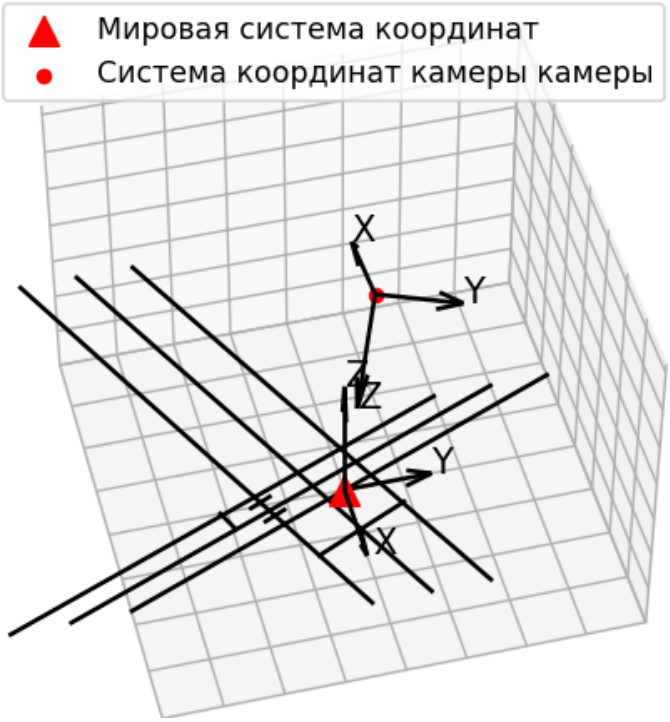

Рисунок 3 – Пример синтетической сцены

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.

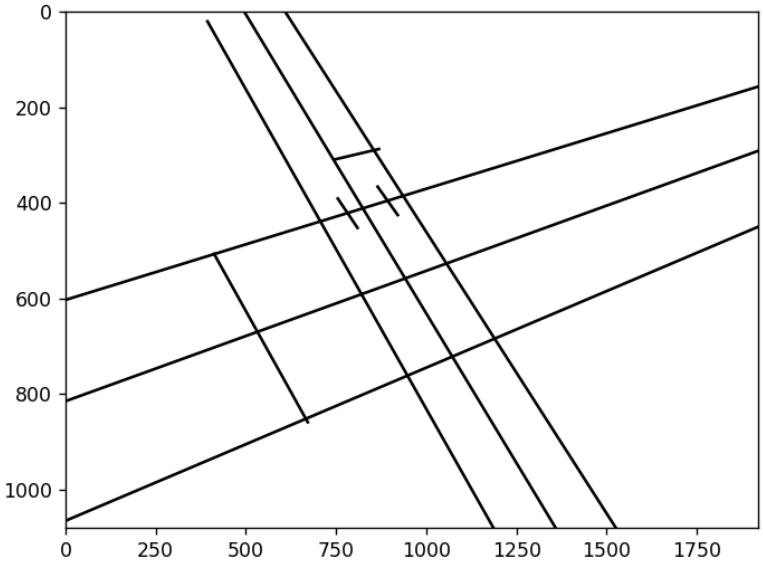

Рисунок 4 – Пример синтетической сцены на плоскости изображения

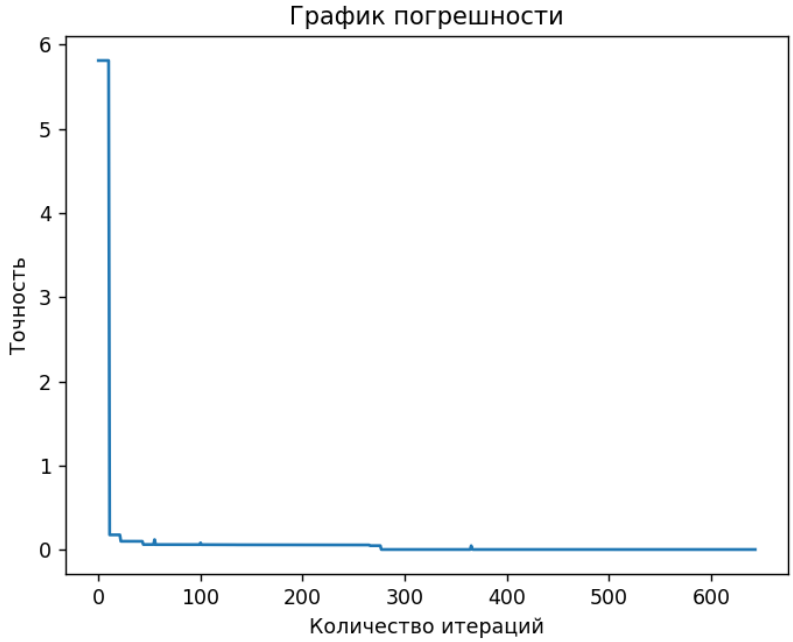

Рисунок 5 – График погрешности целевой функции

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.

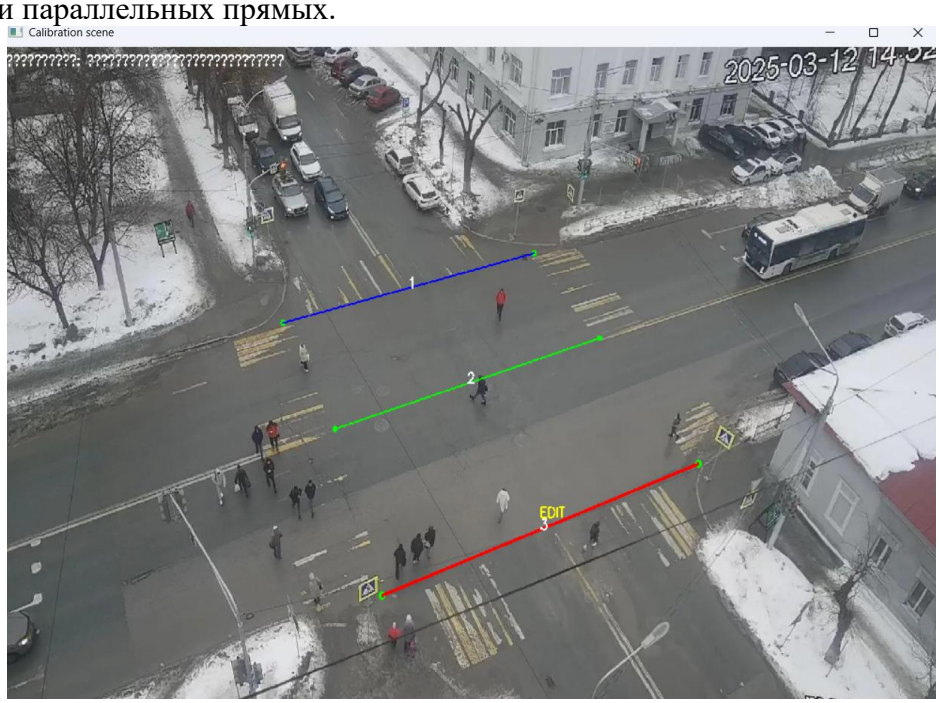

Рисунок 6 – Пример разметки данных через инструмент

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.

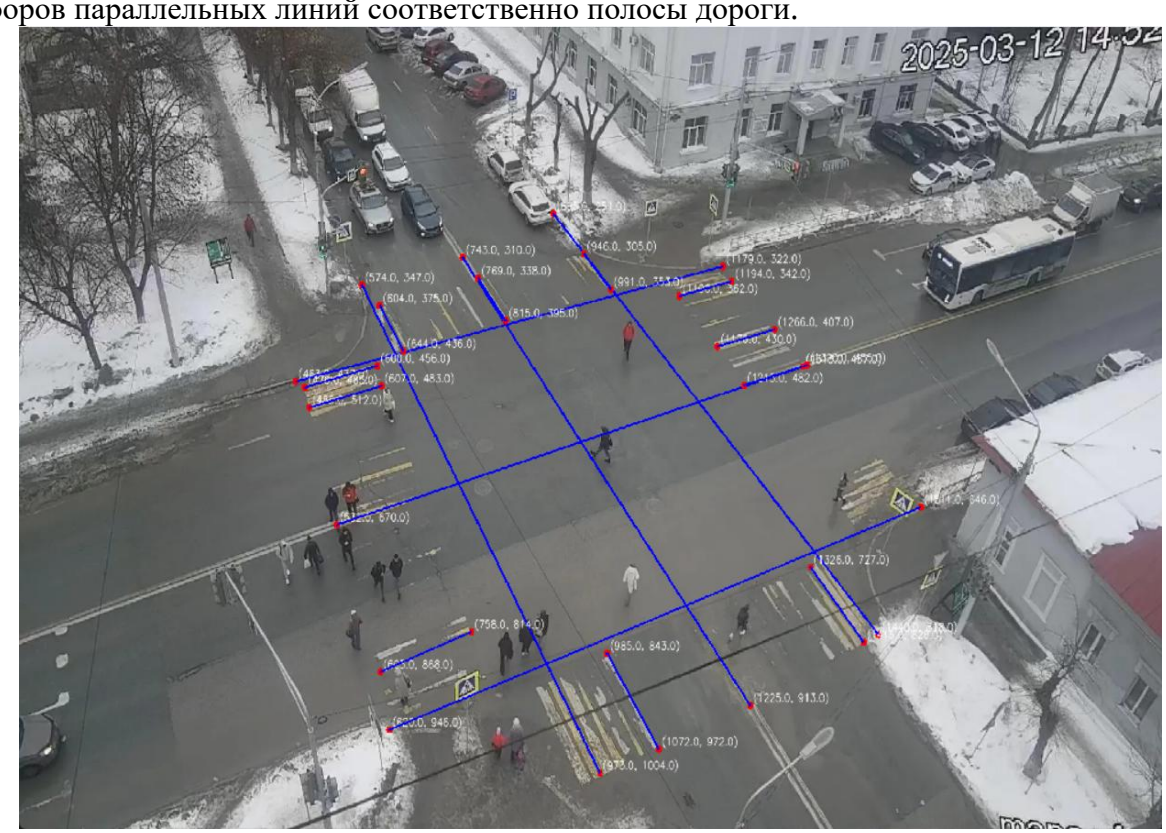

Рисунок 7 – Геометрические ограничения для калибровки камеры с перекрестка Пушкина – Аксакова

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.

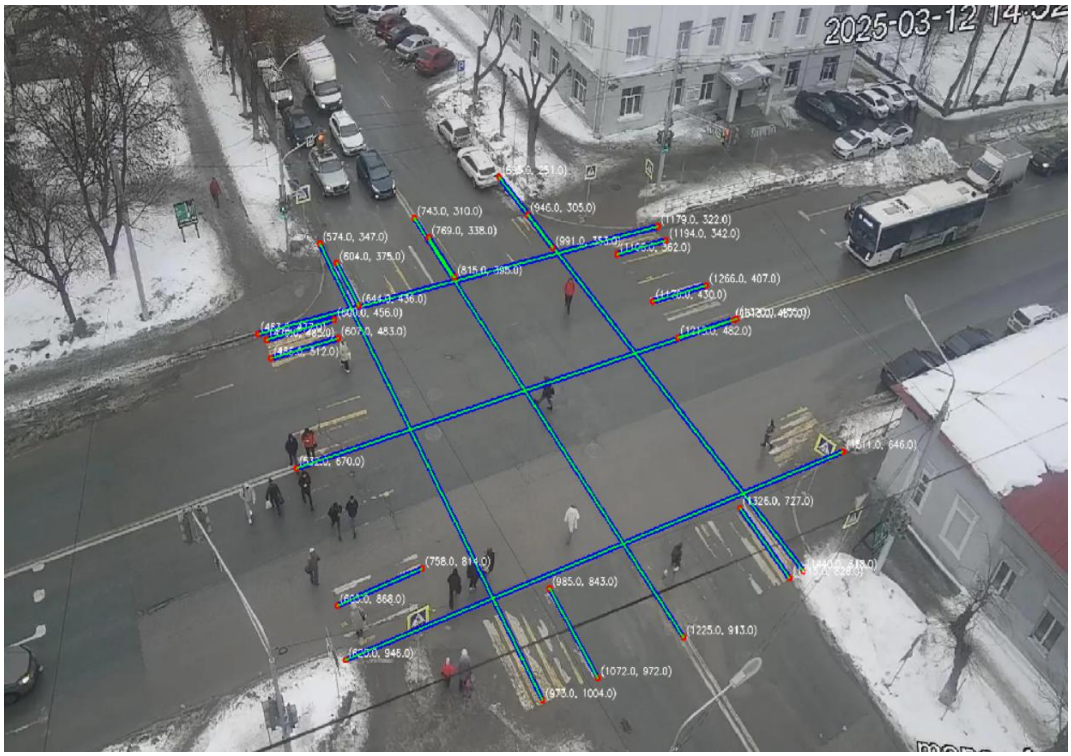

Рисунок 8 - Результат проецирования из координат изображения в мировые координаты и обратно.

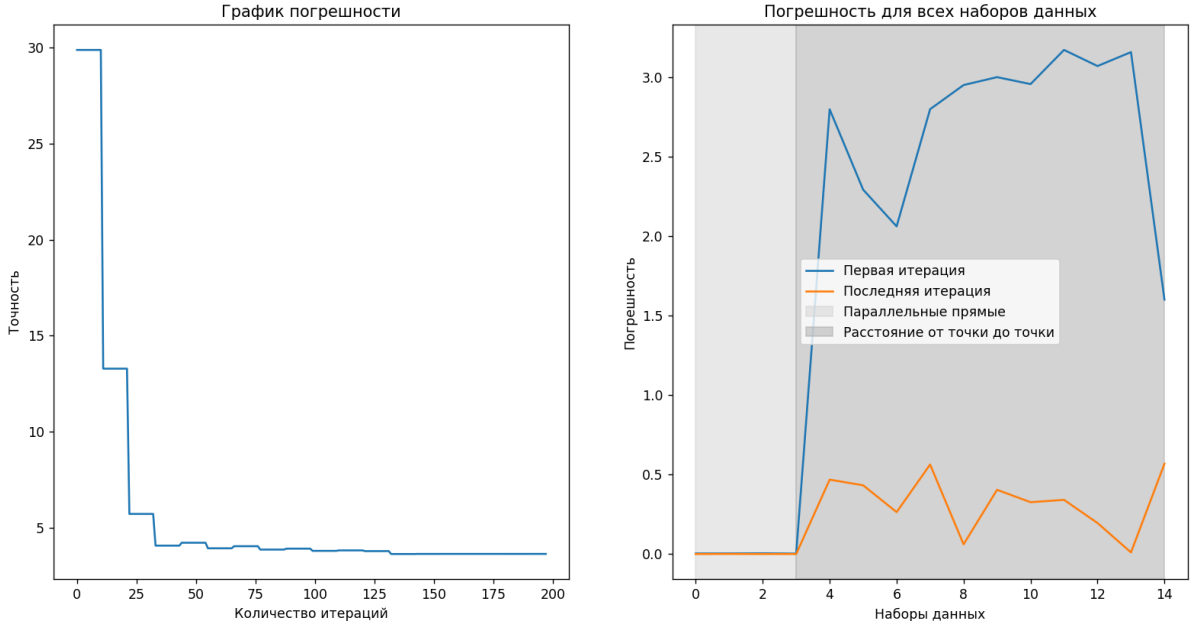

Рисунок 9 – График погрешности целевой функции и погрешности для всех наборов данных на первой и последней итерации.

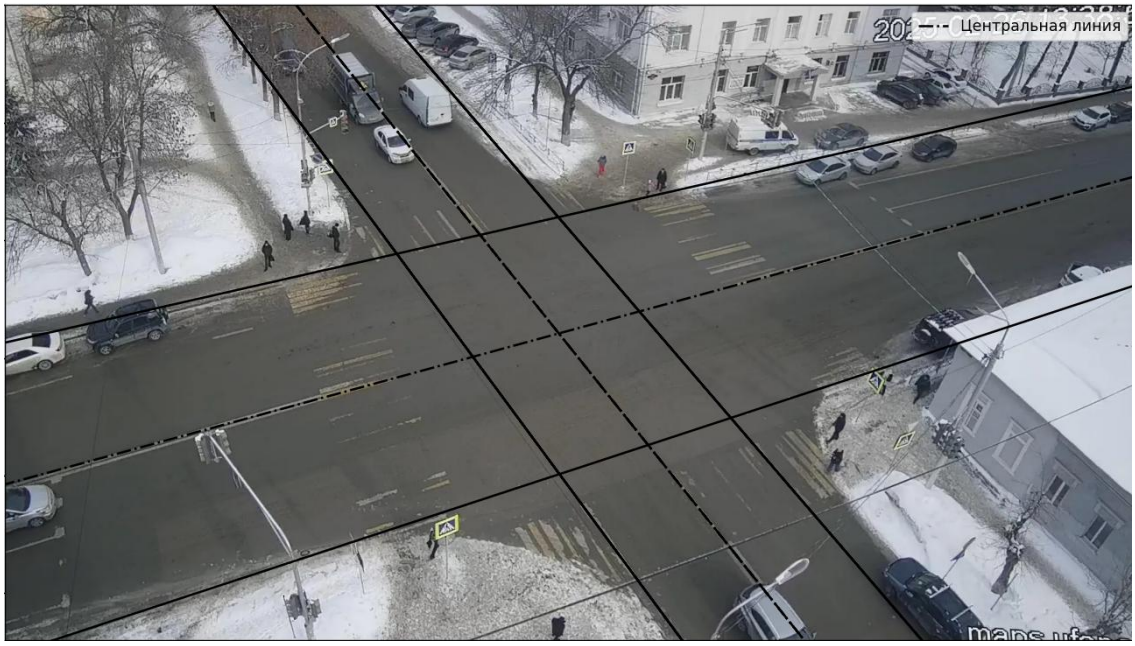

Рисунок 10 – Отображение линий, соответствующих ширине полосы.

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 С
(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()
```