

# ОГЛАВЛЕНИЕ

|   |    |
|---|----|
| ОГЛАВЛЕНИЕ.....   | 5  |
| ВВЕДЕНИЕ.....   | 7  |
| ГЛАВА 1. ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ .....  | 9  |
| 1.1 Обзор систем ориентации и позиционирования объекта .....                            | 9  |
| 1.1.1 Инерциальные системы .....  | 9  |
| 1.1.2 Спутниковые системы.....  | 10 |
| 1.1.3 Системы локального позиционирования .....   | 11 |
| 1.2 Современные методы компьютерного зрения .....                                       | 14 |
| 1.2.1 Метод одновременной навигации и картирования .....                                | 14 |
| 1.2.2 Метод определения структуры объекта в процессе движения.....                      | 16 |
| 1.3 Методы, использующие точки схождения перспективы .....                              | 16 |
| ГЛАВА 2. АЛГОРИТМЫ ГЕОМЕТРИИ ПЕРСПЕКТИВНЫХ<br>ИЗОБРАЖЕНИЙ.....                          | 17 |
| 2.1 Геометрия перспективных изображений.....  | 17 |
| 2.1.1 Методы, основанные на геометрии перспективных изображений ...                     | 18 |
| 2.1.2 Однородные координаты .....   | 26 |
| 2.2 Связь между мировой системой координат и системой координат камеры<br>29            |    |
| 2.3 Модель камеры–обскуры .....   | 32 |
| 2.4 Выделение сегментов линий на изображении методом Джиии.....                         | 36 |
| 2.5 Кластеризация сегментов линий методом Seq. RANSAC .....                             | 43 |
| 2.6 Уточнение ТСП.....  | 47 |
| 2.6.1 Уточнение матрицы поворота камеры относительно мировой<br>системы координат ..... | 49 |
| 2.6.2 Определение ориентации по ТСП .....   | 52 |
| ГЛАВА 3. ПРОЕКТИРОВАНИЕ ПРИКЛАДНОГО ПРОГРАММНОГО<br>ОБЕСПЕЧЕНИЯ.....                    | 54 |
| 3.1 Блок организации запуска приложения и управления тестированием ...                  | 55 |
| 3.2 Управление процессом обработки изображения.....                                     | 60 |
| 3.2.1 Выделение сегментов линий .....   | 62 |
| 3.2.2 Вычисление точек схождения перспективы .....                                      | 66 |
| 3.2.3 Уточнение матрицы поворота на основе полученных направлений<br>ТСП 72             |    |

|  |            |
|--|------------|
| 3.2.4 Вычисление углов Эйлера по известной матрице поворота .....            | 75         |
| 3.3 Блок управления процессом тестирования наборов изображений .....         | 78         |
| <b>ГЛАВА 4. ТЕСТИРОВАНИЕ ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.....</b>       | <b>79</b>  |
| 4.1 Методология разработки и используемые средства.....                      | 79         |
| 4.1.1 Библиотека алгоритмов компьютерного зрения OpenCV .....                | 79         |
| 4.2 Руководство пользователя.....  | 80         |
| 4.2.1 Требования для сборки и работы с приложением.....                      | 80         |
| 4.2.2 Режимы работы и входные аргументы .....                                | 80         |
| 4.2.3 Обычный режим работы.....  | 83         |
| 4.2.4 Режим тестирования набора изображений .....                            | 88         |
| 4.3 Тестирование приложения.....   | 90         |
| 4.3.1 База данных изображений YorkUrbanDB.....                               | 90         |
| 4.3.2 Конфигурация тестирования .....  | 92         |
| 4.4 Результаты тестирования .....  | 92         |
| 4.4.1 Результаты тестирования метода на изображениях «внутри помещения»..... | 92         |
| 4.4.2 Результаты тестирования метода на изображениях «вне помещения»         | 97         |
| 4.4.3 Результаты тестирования на случайных изображениях .....                | 101        |
| <b>ГЛАВА 5. ОРГАНИЗАЦИОННО-ЭКОНОМИЧЕСКАЯ ЧАСТЬ .....</b>                     | <b>105</b> |
| 5.1 Организация и планирование процесса разработки программы .....           | 106        |
| 5.1.1 Техническое задание .....  | 106        |
| 5.1.2 Расчёт стоимости проекта .....   | 106        |
| 5.1.3 Затраты на выплату исполнителям .....                                  | 109        |
| 5.1.4 Определение количества исполнителей.....                               | 119        |
| 5.1.5 Календарный график выполнения работ .....                              | 119        |
| 5.2 Расчёт сметы затрат .....  | 121        |
| 5.2.1 Суммарные затраты.....   | 124        |
| <b>ЗАКЛЮЧЕНИЕ .....</b>  | <b>125</b> |
| <b>СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ.....</b>                                   | <b>127</b> |
| <b>ДОПОЛНИТЕЛЬНЫЕ ИЛЛЮСТРАЦИИ.....</b>                                       | <b>131</b> |
| <b>ПРИЛОЖЕНИЕ 1. ДИАГРАММА ГАНТА ВЫПОЛНЯЕМЫХ РАБОТ .....</b>                 | <b>135</b> |

# ВВЕДЕНИЕ

Последние десятилетия наблюдается высокий уровень интереса к автономной (автопилотируемой) технике. Автопилотируемым системам можно найти применение как в условиях экстремальных для человека ситуаций, так и при выполнении шаблонных работ, в том числе при организации типовых маршрутов пассажиро- и грузоперевозок. В числе наиболее проблемных подсистем самоуправляемой техники значится система навигации, и, в частности, подсистема позиционирования. К основным проблемам таких систем относят дрейф — рост погрешностей измерения в процессе работы. Системы, принцип функционирования которых не допускает дрейфа, оказываются аппаратнозатратными, и, как следствие, дорогими (см. 1.1).

К системам не имеющим дрейфа и не требующим значительных затрат на оборудование, относят системы машинного (компьютерного) зрения (см. 1.2).

Одна из подзадач, возникающих при построении системы позиционирования с использованием алгоритмов компьютерного зрения, — определение углов наклона и поворота видеокамеры, установленной на перемещающемся объекте. В этом случае, видеокамера, позиционируя движение объекта, выполняет роль гироскопа.

В 2012 году Роберт Пише и его ученик Вилле Хуттунен предложили и апробировали в помещении метод определения трехмерной ориентации монокулярной камеры с использованием точек схождения перспективы (ТСП), обнаруженных в последовательностях изображений [1].

Разработчики метода выделяют следующий набор его достоинств [1]:

- наличие большого количества линейных объектов правильной геометрической формы позволяет достигнуть высокой точности;
- число интересующих нас ТСП ограничено — не больше трех;
- ТСП не зависят от положения камеры — только от ее ориентации;

- работность относительно случайных нестационарных объектов, попадающих в кадр (люди, транспортные средства и т.п.).

К недостаткам этой работы можно отнести тот факт, что тестирование метода проводилось в камеральных условиях (только в помещении). Кроме того, применялась калиброванная камера мобильного телефона Nokia N900, имеющая имеющая внутренний инерциальный измерительный модуль (ИИМ) с трехосным акселерометром, трехступенчатым гироскопом и магнитометром. Наличие такого регистрирующего устройства по сути способствовало обеспечению точности позиционирования объекта.

Кроме того, авторы метода указывают в качестве недостатка снижение точности позиционирования при небольшом количестве характерных линий, выделяемых в кадре.

Целью настоящей работы является апробация метода Хуттунена–Пише и проверка применимости подхода в условиях городской застройки, а также — при ориентации среди объектов живой природы.

Предполагается, что видеоизображение предварительно декомпозировано на серию кадров (операция производится автоматически при съемке видеокамер современных гаджетов). Серии по 25 кадров с разрешением не хуже 720 точек на дюйм можно получать камерами Nokia N900, смартфонами и планшетами Samsung GalaxyTAB III и выше и т.п.

В качестве набора тестовых данных метода выбраны фотографии из базы изображений YorkUrbanDb [2], созданной на базе Centre for Vision Research Йоркского университета города Торонто.

Для оценки точности позиционирования вне помещений использовался набор выбранных случайным образом изображений, находящихся в открытом доступе.

В первой главе подробно раскрывается проблематика, описаны существующие аналоги, как с использованием методов компьютерного зрения,

так и без него. Во второй главе описан метод Хуттунена–Пише и алгоритмы, обеспечивающие его реализацию. Третья глава содержит описание структуры тестового приложения, реализующего рассматриваемый метод. Четвертая глава посвящена решению проблем технологического характера, возникших при реализации проекта. В заключительной, пятой главе, проводится организационно-экономическое обоснование разработки прикладного программного обеспечения.

Данная работа содержит 120 страниц, 40 иллюстраций, 22 таблицы и 28 библиотечных ссылок. Графическая часть проекта приводится в презентации (схемы, диаграммы, графики, иллюстративный материал — 11 слайдов).

## ГЛАВА 1. ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ

### 1.1 *Обзор систем ориентации и позиционирования объекта*

С целью ориентации и навигации автономного движущегося объекта в настоящее время широко используются три варианта решения: инерциальные системы, спутниковые системы и системы локального позиционирования. Кратко рассмотрим каждое из решений.

#### 1.1.1 **Инерциальные системы**

Среди популярных применяемых в автономной робототехнике решений — использование инерциальных систем навигации (ИНС). Подобные системы содержат набор акселерометров для определения параметров линейного ускорения, а также гироскоп (или акселерометры, измеряющие центробежное ускорение) для определения углов поворота и наклона. На основе данных этих датчиков производится последующее вычисление вектора скорости и координат объекта (рис. 1).

Преимущества методов инерциальной навигации и ориентации состоят в автономности, помехозащищенности и возможности полной автоматизации

всех процессов навигации [3]. Основная же проблема — наличие дрейфа, то есть накопление ошибки со временем работы. Различными техниками можно уменьшить величину ошибки, но не избавиться вовсе.



Рисунок 1 — Схема работы ИНС, рисунок заимствован из [3].

### 1.1.2 Спутниковые системы

Современная спутниковая навигация основывается на использовании принципа беззапросных дальномерных измерений между навигационными спутниками и потребителем. Это означает, что потребителю передается в составе навигационного сигнала информация о координатах спутников. Одновременно (синхронно) производятся измерения дальностей до навигационных спутников. Способ измерений дальностей основывается на вычислении временных задержек принимаемого сигнала от спутника по сравнению с сигналом, генерируемым аппаратурой потребителя (рис. 2) [4].

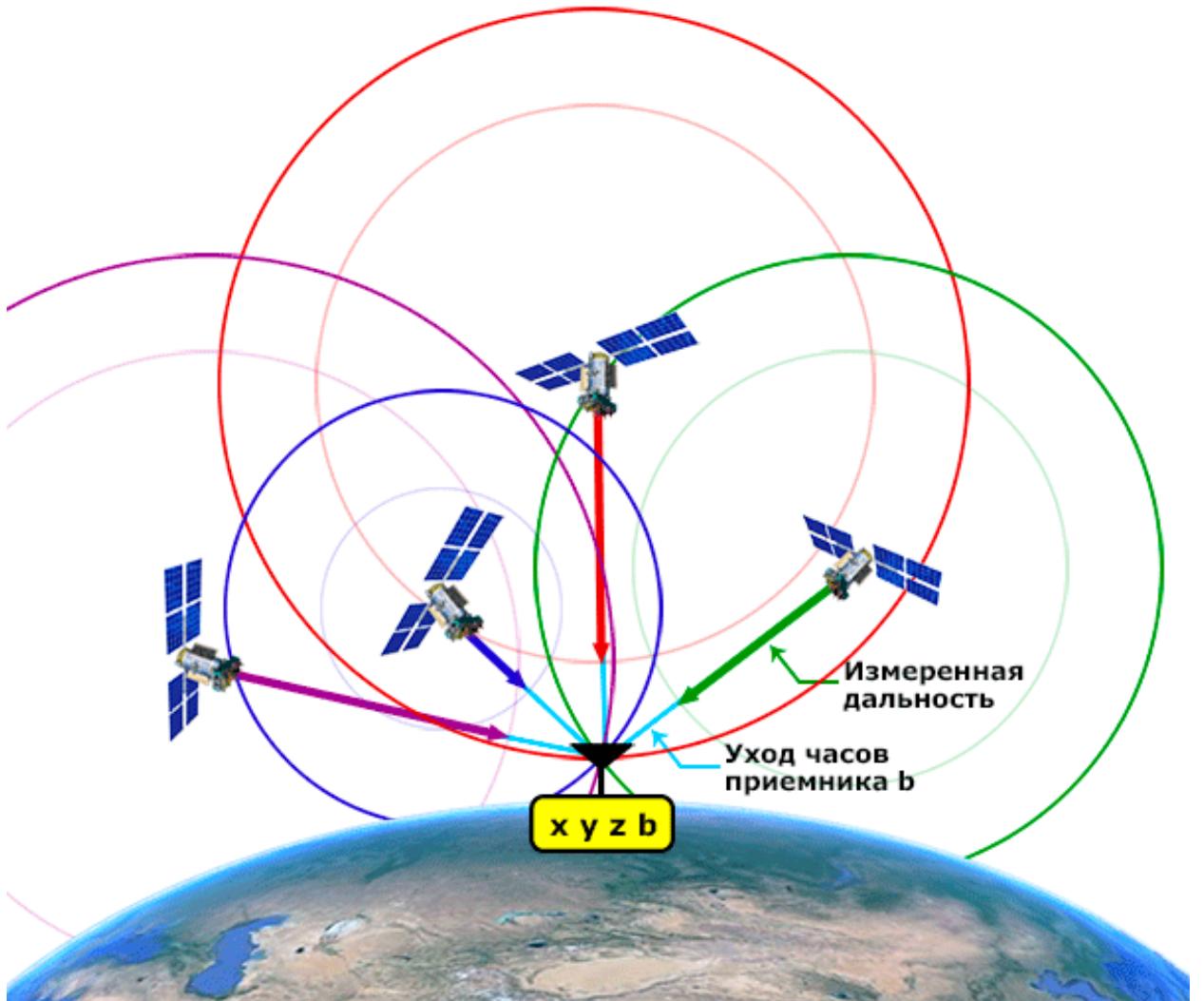


Рисунок 2 — Схема принципа работы спутниковых систем навигации, рисунок заимствован из [4].

Достоинством спутниковых систем является возможность глобального позиционирования при отсутствии дрейфа.

Недостатком спутниковых систем глобального позиционирования является невозможность локализации при неустойчивом спутниковом сигнале. К тому же использование дополнительного оборудования (GPS-приемников) делает этот метод достаточно затратным.

### 1.1.3 Системы локального позиционирования

В ряде практических задач глобальное позиционирование оказывается не актуальным, предпочтение отдается системам локального позиционирования.

Среди систем локального позиционирования распространено использование инфракрасных или ультразвуковых датчиков.

Подобные системы обычно требуют наличия нескольких опорных приемников, относительно которых вычисляется месторасположение объекта–носителя передатчика (неявное задание системы координат).

Преимуществом систем этого класса является достаточно высокая точность локализации объекта. К недостаткам можно отнести высокую стоимость и значительное количество условий и ограничений применимости метода.

Если взять в качестве примера, локальную систему LiDAR (рис. 3) одного из лидеров индустрии — компании Velodyne, оборудованием которой оснащены самоуправляемые автомобили и другие транспортные средства компаний Google, Toyota, Caterpillar, Ford, Lockheed Martin, Oshkosh, GM Nissan, то ее стоимость варьируется от 8 тысяч долларов за самую дешевую 16–канальную систему, до 80 тысяч долларов — за 64–канальную [5].

Также тип используемого сигнала накладывает следующие серьезные ограничения на использование: расстояние до приемников не больше 10 метров, отсутствие препятствий, отражений и помех. К тому же такие системы пригодны только на заранее подготовленной (там, где установлены опорные вышки — приемники) территории.

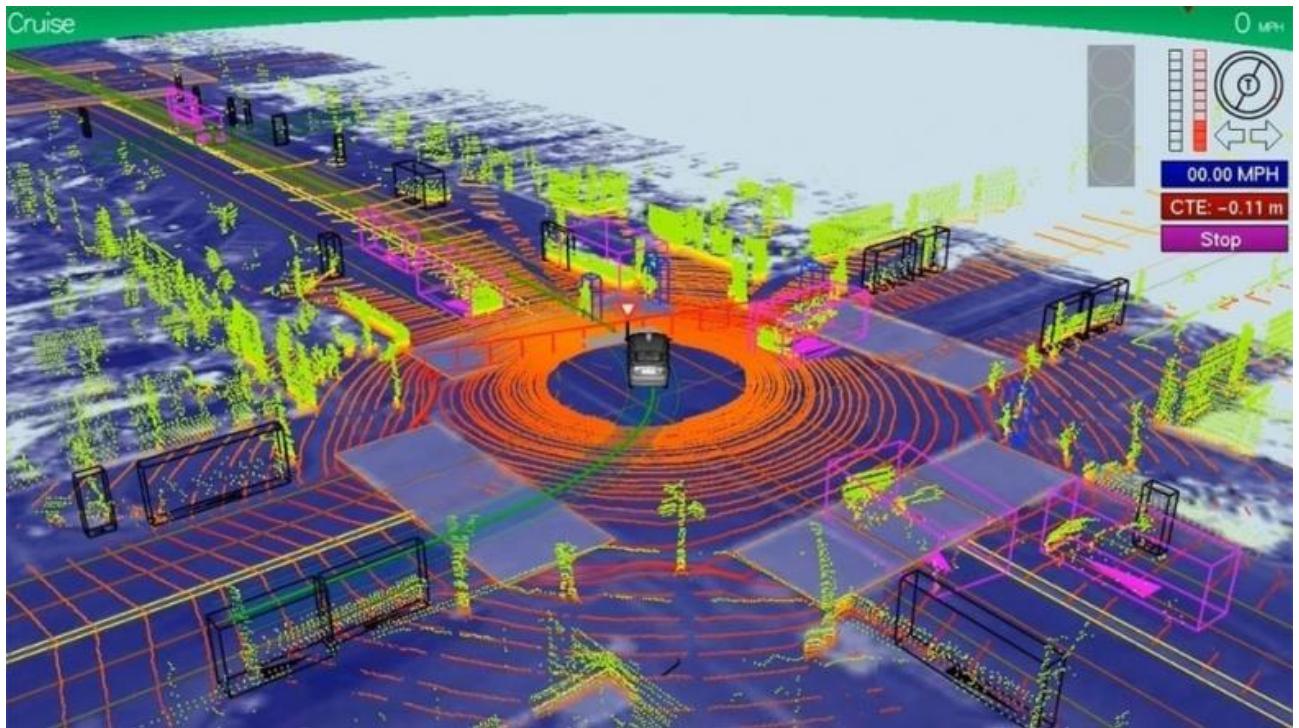


Рисунок 3 — Пример работы системы позиционирования LiDAR, работающей на автономных автомобилях исследовательского центра компании Google, рисунок заимствован из [6].

Этого недостатка лишены системы, осуществляющие построение трехмерной карты окружающей среды на основе данных эхолокации. Для эхолокации обычно используются инфракрасные, ультразвуковые и лазерные датчики. К недостаткам таких систем можно отнести очень высокую стоимость оборудования. Среди подобных систем набирает популярность использование методов компьютерного зрения.

Современной альтернативой дорогим системам локального позиционирования выступают системы, применяющие методы машинного (компьютерного) зрения. В качестве аппаратного обеспечения таких систем используют видеодатчики, имеющие незначительный размер, и как следствие – низкое энергопотребление и невысокую стоимость. Подобные системы на основе анализа изображений не имеют дрейфа, при этом может достигаться довольно высокая точность, сравнимая с ИНС потребительского класса [1].

На сегодняшний день выделяют следующие перспективные методы компьютерного зрения:

- метод одновременной навигации и картирования (simultaneous localization and mapping, SLAM);
- метод определения структуры объекта в процессе движения (structure from motion, SfM);
- методы, использующие точки схождения перспективы (метод Хуттунена–Пише).

## **1.2 Современные методы компьютерного зрения**

Большинство из предлагаемых методов основано на обнаружении базисных элементов изображения и слежения за ними в потоке изображений. Среди распространенных — метод одновременной навигации и картирования (simultaneous localization and mapping, SLAM), а также метод определения структуры объекта в процессе движения (structure from motion, SfM) [1]. Оба этих метода в той или иной степени пытаются построить двух- или трехмерную модель окружающей среды, относительно которой происходит движение камеры.

### **1.2.1 Метод одновременной навигации и картирования**

На рис. 4 приведен пример реализации метода одновременной навигации и картирования, предполагающего построения двухмерной модели окружающей среды, относительно которой происходит движение камеры.



Рисунок 4 — Пример построенной карты местности в процессе движения робота с использованием SLAM–метода [7].

Обычно в SLAM методах используются расширенный фильтр Калмана, фильтр частиц (последовательный метод Монте–Карло) и сканирование с сопоставлением для получения оценки функции апостериорной вероятности для позиции робота и параметров карты. Для повышения качества реконструкции карты окружающей среды и оценки позиционирования камеры метода корректировки пучков (bundle adjustment), метод принимает в расчет полученные оценки положения и позиции ориентиров, а также параметры относительного движения и оптические характеристики используемой камеры [7].

Очевидным недостатком метода является двумерная визуализация результата, что не дает, например, в поставленной задаче оценить углы ориентации камеры.

### 1.2.2 Метод определения структуры объекта в процессе движения

Для SfM–методов характерно использование детекторов особенностей, таких как scale–invariant feature transform (SIFT) и speeded up robust features (SURF).

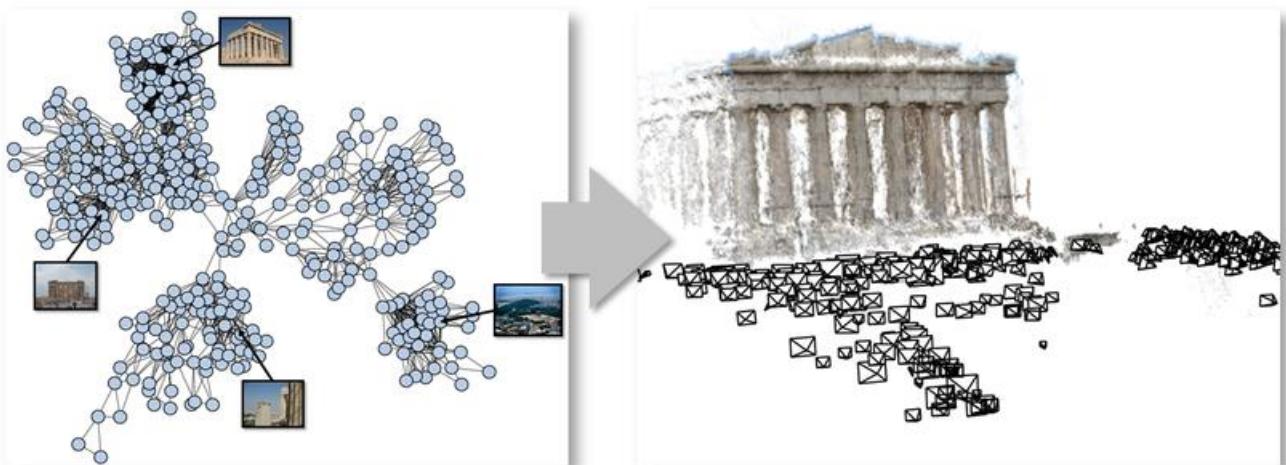


Рисунок 5 — Пример воссозданных трехмерных наблюдаемых объектов с использованием SfM–метода [7].

На рис. 5 приведен результат работы метода в виде воссоздаваемых трехмерных объектов, наблюдаемых камерой.

Очевидно, этот метод относится к узкоспециализированным и имеет возможность локализации движущегося объекта (робота, транспортного средства) только при наличии цифровой карты местности и программного обеспечения, реализующего распознавание объекта.

### 1.3 Методы, использующие точки схождения перспективы

Следует отметить, что ряд практических приложений допускает разделение задач навигации и ориентации (позиционирования) объекта, что позволяет снизить стоимость решения при удовлетворительной точности. В частности, при построении системы позиционирования, может оказаться достаточным определение углов наклона и поворота видеокамеры,

установленной на объекте. В этом случае, видеокамера выступает в виде аналога гироскопа, а углы могут быть вычислены с использованием точек схождения перспективы (ТСП), полученных по серии изображений.

Алгоритм обработки изображений в рамках метода Хуттунена-Пише [1] выглядит следующим образом:

- регистрация изображения или видеоряда с последующей декомпозицией его на кадры;
- чтение изображения и обнаружение на нем сегментов линий (СЛ);
- последовательное выделение трех наибольших кластеров СЛ, индуцируемых точками схождения перспективы (ТСП), в предположении, что они образуют три взаимно ортогональных направления;
- уточнение направлений ТСП;
- вычисление углов ориентации по полученным направлениям ТСП.

## ГЛАВА 2. АЛГОРИТМЫ ГЕОМЕТРИИ ПЕРСПЕКТИВНЫХ ИЗОБРАЖЕНИЙ

### 2.1 *Геометрия перспективных изображений*

Теория точек схождения перспективы (ТСП) рассматривается обычно в терминах проективной геометрии, изучающей геометрические свойства, являющихся инвариантами относительно проективных преобразований, а также сами эти преобразования. Проецирование трехмерной сцены на двухмерную плоскость изображения, осуществляемое фото- или видеокамерой, — одно из таких преобразований.

Одной из интересующих нас особенностей проективного преобразования является тот факт, что параллельность прямых не является инвариантом

относительно него.

### 2.1.1 Методы, основанные на геометрии перспективных изображений

В разделе будут рассмотрены различные приложения геометрии перспективных изображений.

#### а) метод Хуттунена–Пише

Метод Хуттунена–Пише заключается в определения трехмерной ориентации моноокулярной камеры с использованием точек схождения перспективы (ТСП), обнаруженных на изображениях. Использование только ТСП делает слежение за ориентацией объекта значительно проще и быстрее по следующим причинам [1]:

- ТСП порождаются свойствами линейных объектов, которые различимы и, предположительно, доминируют на изображениях внутри помещений и в условиях плотной городской застройки, где использование глобальных (спутниковых) систем навигации затруднено.
- Число интересующих нас ТСП ограничено (не больше трех).
- ТСП не зависят от положения камеры — только от ее ориентации.
- Методы на основе поиска ТСП являются робастными относительно случайных нестационарных объектов, попадающих в кадр (люди, транспортные средства и т.п.).

Данный метод использует алгоритм детектирования сегментов линий (СЛ) на изображении, предложенный Джии и др. [8] (рис. 6).

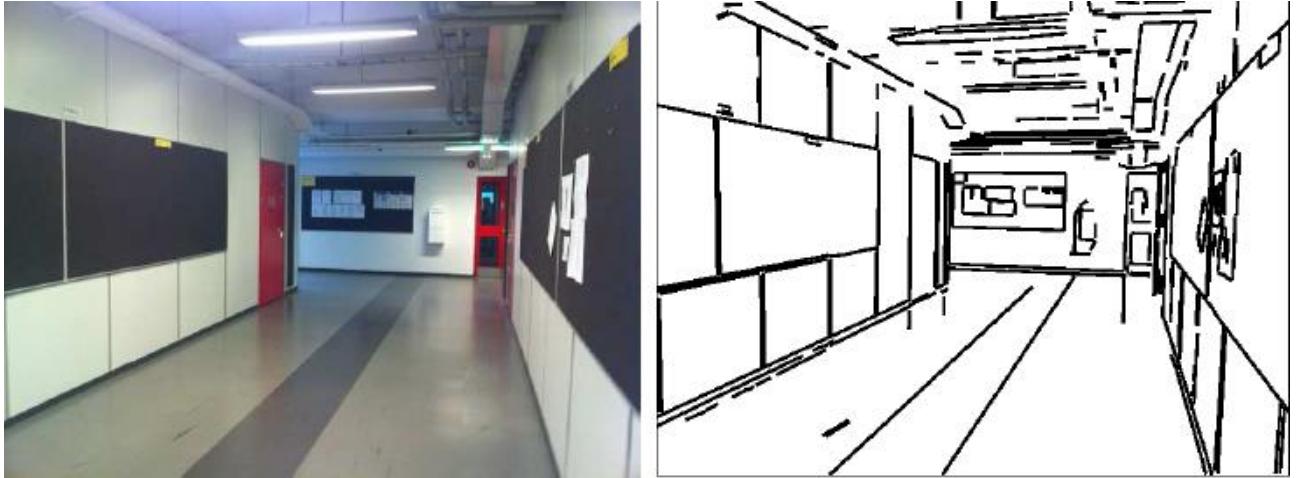


Рисунок 6 — Пример результата работы детектора сегментов линий Джиии. Слева: оригинальное изображение; справа: обнаруженные сегменты линий. Рисунок заимствован из [1]

Среди полученного набора СЛ выделяются три наибольших кластера, задающих линии, сходящихся в трех точках, называемых ТСП. Данный этап проводится с использованием адаптивного алгоритма RANSAC [9].

Полученные три кластера СЛ позволяют получить грубые оценки ТСП. Предполагается, что найденные ТСП задают три взаимно перпендикулярных направления трехмерного евклидова пространства сцены (рис. 7).

На рис. 7 слева сверху: все обнаруженные на изображении СЛ; остальные: кластеры трех доминирующих ТСП, которым соответствуют три взаимно перпендикулярных направления.

Затем грубые оценки ТСП улучшаются. Для начала координаты сегментов линий каждого кластера переводятся из координат изображения, выраженных в пикселях, в нормализованные координаты в метрической системе как проекции точек в системе координат камеры на плоскость  $Z = 1$ . Такое преобразование осуществляется домножением координат точек слева на матрицу обратную матрице калибровки. Затем для каждой из ТСП приближенно решается переопределенная СЛАУ, которая обозначает связь между линиями, проходящими через СЛ соответствующего кластера — линии должны быть взаимно параллельны. Задача минимизации вектора невязки решается с

использованием сингулярного разложения матрицы [37].

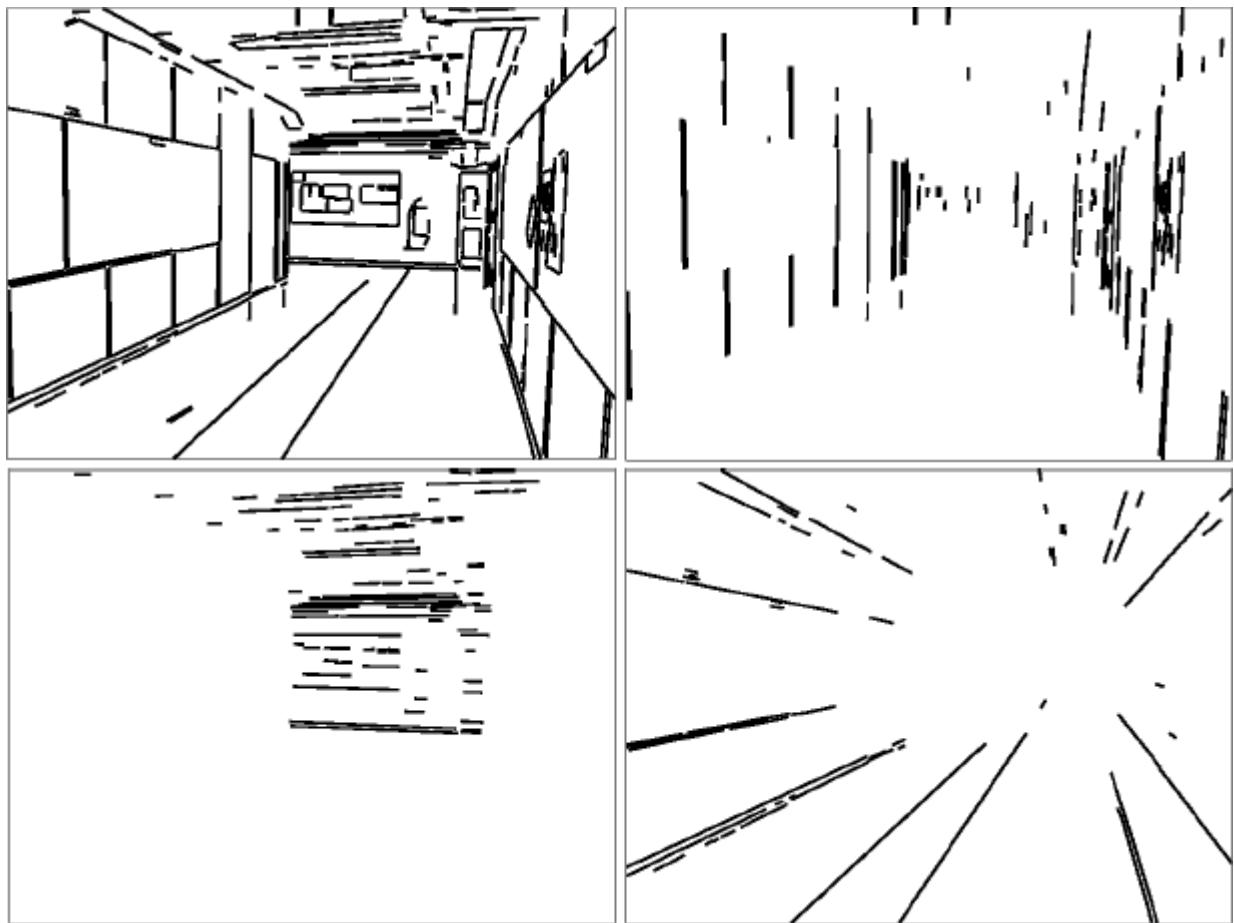


Рисунок 7 — Пример результата кластеризации алгоритмом RANSAC, рисунок заимствован из [1].

Полученные векторы направлений ТСП исследуются на взаимную ортогональность. Для получения углов ориентации камеры достаточно хотя бы двух правильно найденных взаимно ортогональных направлений ТСП, так как оставшееся третье направление может быть вычислено как векторное произведение двух имеющихся. На данном этапе отбрасываются заведомо ложные полученные направления и довоычисляются оставшиеся по описанному выше алгоритму.

В итоге имеется система трех приближенно взаимно ортогональных направлений. Для дальнейшего определения ориентации камеры данная система требует дополнительной ортогонализации. Данная задача решается с помощью сингулярного разложения матрицы, составленной из векторов

направлений найденных ТСП [10].

Полученная ортонормированная система векторов задает матрицу поворота камеры, по которой можно вычислить ее углы ориентации (рис. 8).



Рисунок 8 — Пример результата работы созданной в рамках работы реализации метода на изображении городской местности.

На рис. 8 слева: оригинальное изображение; справа: результат, где цветами промаркованы кластеры СЛ, относящиеся взаимно ортогональным ТСП. Полученные углы ориентации: крен  $\cong 2$  градуса, тангаж  $\cong -27$  градусов, рыскание  $\cong -1$  градус.

#### б) другие применения методов поиска точек схождения перспективы

Помимо использования ТСП в приложениях навигации автономных транспортных средств существуют и другие применения данных методов. Подобные методы могут применяться в трехмерной реконструкции сцены, в частности, в реконструкции архитектурных видов [11, 12], для корректировки фотографий или для определения внутренних параметров камеры, то есть для ее калибровки [13, 14, 15, 16].

Существует довольно большое число аналогичных работ, посвященных поиску точек схождения перспективы, для решения различных задач. Данные работы можно условно разделить на три категории [17]. Первые две требуют

знания внутренних параметров камеры, а третья работает в условиях некалиброванной камеры.

В качестве детектора сегментов линий независимо от категории метода обычно используются методы, основанные на преобразовании Хафа или анализе связанных компонент ориентаций градиента изображения. Среди последних наиболее популярен так называемый Кэнни–детектор (Canny edge detector) [18]. Довольно новым и только начинающим набирать популярность является метод, предложенный Джииои [8]. Он относится к методам того же типа, что и Кэнни–детектор, но отличается более высокой скоростью и качеством работы, однако еще не получил широкого распространения, так как не имеет реализаций в популярных пакетах и фреймворках, используемых в компьютерном зрении.

Имея внутренние параметры камеры, линии на изображении вместе с конечными и бесконечными ТСП могут быть представлены двумерными нормализованными однородными векторами, то есть единичными векторами на Гауссовой сфере с центром в оптическом центре камеры. Бэрнард и др. предложили использование преобразования Хафа на квантованной сфере Гаусса, однако позднее было обнаружено, что это часто приводит к нахождению ложных ТСП [18]. С тех пор большинство работ основываются на параллельности или ортогональности преобладающих на изображении структур с целью исключить ложные срабатывания.

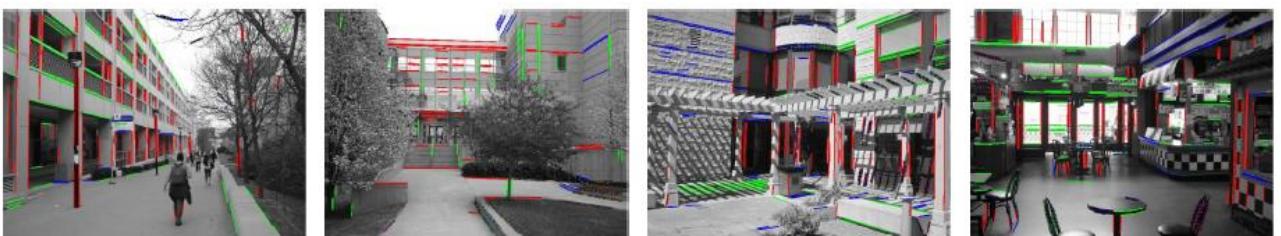


Рисунок 9 — Пример результатов работы определения ТСП в работе Тардиффа, рисунок заимствован из [18].

Более поздние методы, использующие имеющуюся информацию о

внутренних параметрах камеры, основаны на так называемом «предположении Манхэттенского мира» (Manhattan world assumption), что преобладающие на сцене структуры взаимно перпендикулярны [19]. Алгоритмы данного типа решают задачу оценки векторов так называемых Манхэттенских направлений (направлений трех взаимно перпендикулярных ТСП), что эквивалентно определению ориентации камеры (рис. 9). Разработаны методы данного типа, основанные на использовании или градиентов сегментов линий, или самих СЛ [20, 18], найденных на изображении.

Методы последней категории не предполагают никаких знаний о внутренних параметрах камеры и своей задачей обычно ставят оценку не взаимно ортогональных в общем случае направлений. Подобные методы особенно подходят для калибровки камеры или восстановления трехмерной структуры фасадов строений.

Среди имеющихся решений — использование принципа Гельмгольца, который декларирует, что любое сильное отклонение от однородного шума на изображении может быть различимо и выделено. В данном случае «сильные отклонения» ни что иное как геометрические примитивы: прямые и кривые линии, замкнутые кривые, многоугольники, локальные группы точек или примитивов. Принцип Гельмгольца используется для разбиения изображения на значимые регионы перспективы (Meaningful vanishing regions) совместно с принципом минимальных длин дескрипторов (Minimal description length, MDL) для отсечения ложных ТСП [21].

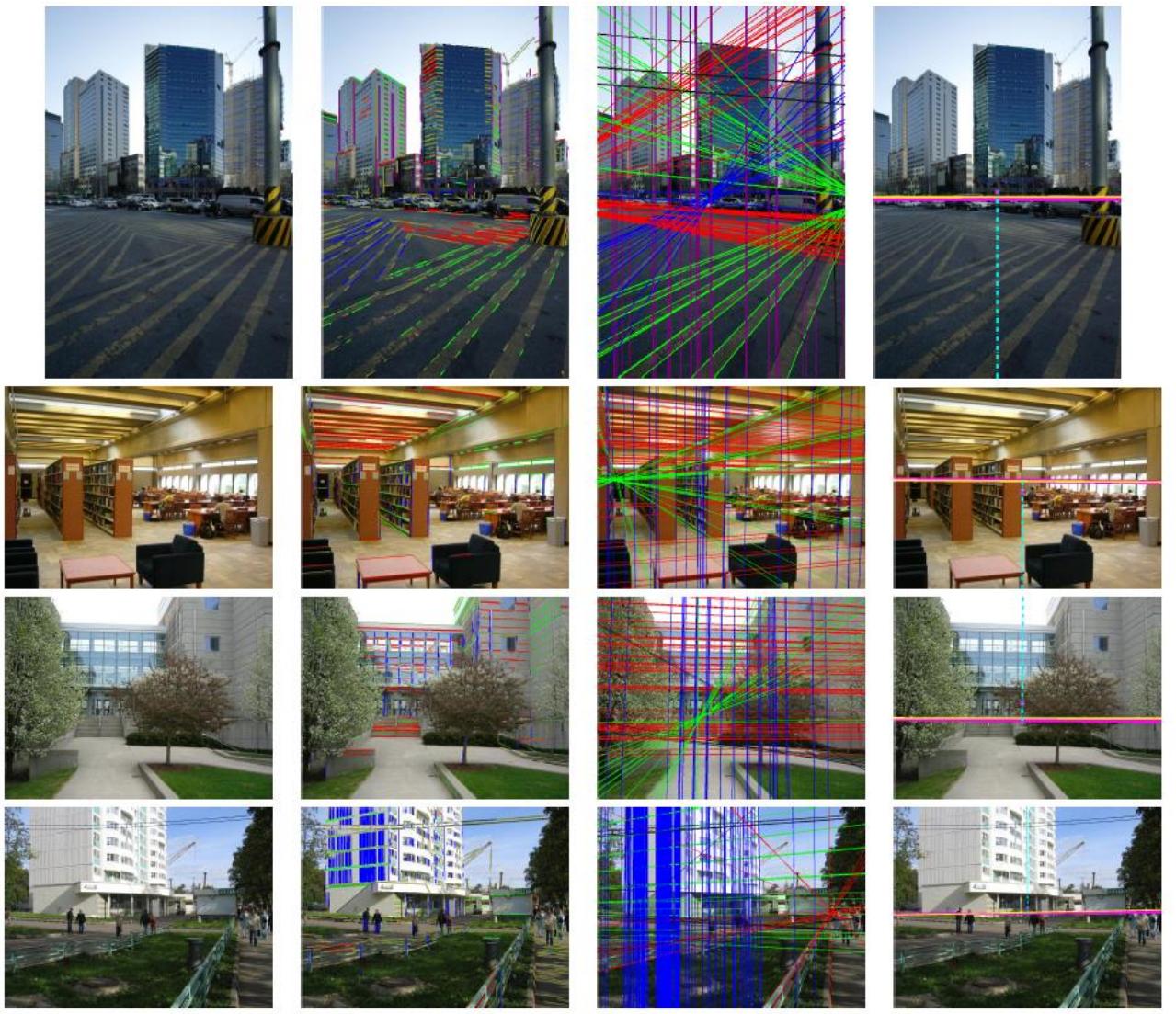


Рисунок 10 — Пример результата работы метода, предложенного Третьяковой и др. [21].

Для каждого входного изображения на рис. 10 (крайние слева) слева направо приведены полученные: сегменты линий, кластеры линии, сгруппированные по направлениям, результаты оценки горизонта и зенита.

Другой способ предполагает использование алгоритма максимизации ожидания (Expectation Maximization, EM) для структур, представленных на Гауссовой сфере, который расширяем на случай некалиброванной камеры. Данный подход с использованием EM–оптимизации требует предвычислений грубых оценок ТСП [18]. Популярным решением данной задачи, как и в случае с методом Хуттунена–Пише, является кластеризация СЛ по их ориентации и вычисление ТСП для выделенных кластеров. Обычно для таких целей

используется алгоритм RANSAC.

Так как количество интересующих направлений ТСП больше одного, выделение кластеров можно проводить по-разному. В случае алгоритма RANSAC существуют две модификации оригинального метода, решающего задачу выделения единственной модели в условиях шума.

Одна из них называется последовательным алгоритмом RANSAC (sequential RANSAC, seq. RANSAC) и заключается в поочередном выделении моделей, когда каждая новая выделенная модель удаляется из рассматриваемого множества. Такой подход сохраняет высокую скорость оригинального метода, но не является оптимальным.

Другая модификация является оптимальной в случае отсутствия пересечений моделей и предполагает одновременный поиск моделей и называется multiRANSAC. Минусом модификации является необходимость задания точного количества искомых моделей, которое заранее зачастую неизвестно.

Помимо алгоритма RANSAC задача выделения направлений ТСП иногда решается с использованием метода, основанного на случайном преобразовании Хафа (Randomized Hough Transform, RHT) с последующим поиском пиков на гистограмме, полученной по случайной выборке минимального набора СЛ, в некотором дискретном пространстве. Несмотря на то, что RHT не требует знания о количестве искомых моделей, данный метод страдает низкой точностью и быстродействием. Особенно плохо метод работает на сильно зашумленных данных.

Относительно новым и перспективным является также метод J-Linkage, предложенный Тольдо и Фузиелло [22]. Данный метод призван решить проблему обоих методов будучи быстрым и оптимальным (рис. 11). В отличие от алгоритма RANSAC метод J-Linkage подходит к проблеме поиска моделей с обратной стороны — рассматривается не распределение отклонения точек данных относительно гипотез, а распределение отклонения гипотез

относительно точек. В качестве пояснения, в терминах решаемой в данной работе задачи кластеризации сегментов линий, в соответствии с J-Linkage для каждого сегмента линии (или случайной выборки из общего числа) проводилось бы исследование распределения отклонений потенциальных точек схождения перспективы, и принадлежность СЛ к кластеру определялась бы именно на основе анализа данного распределения.

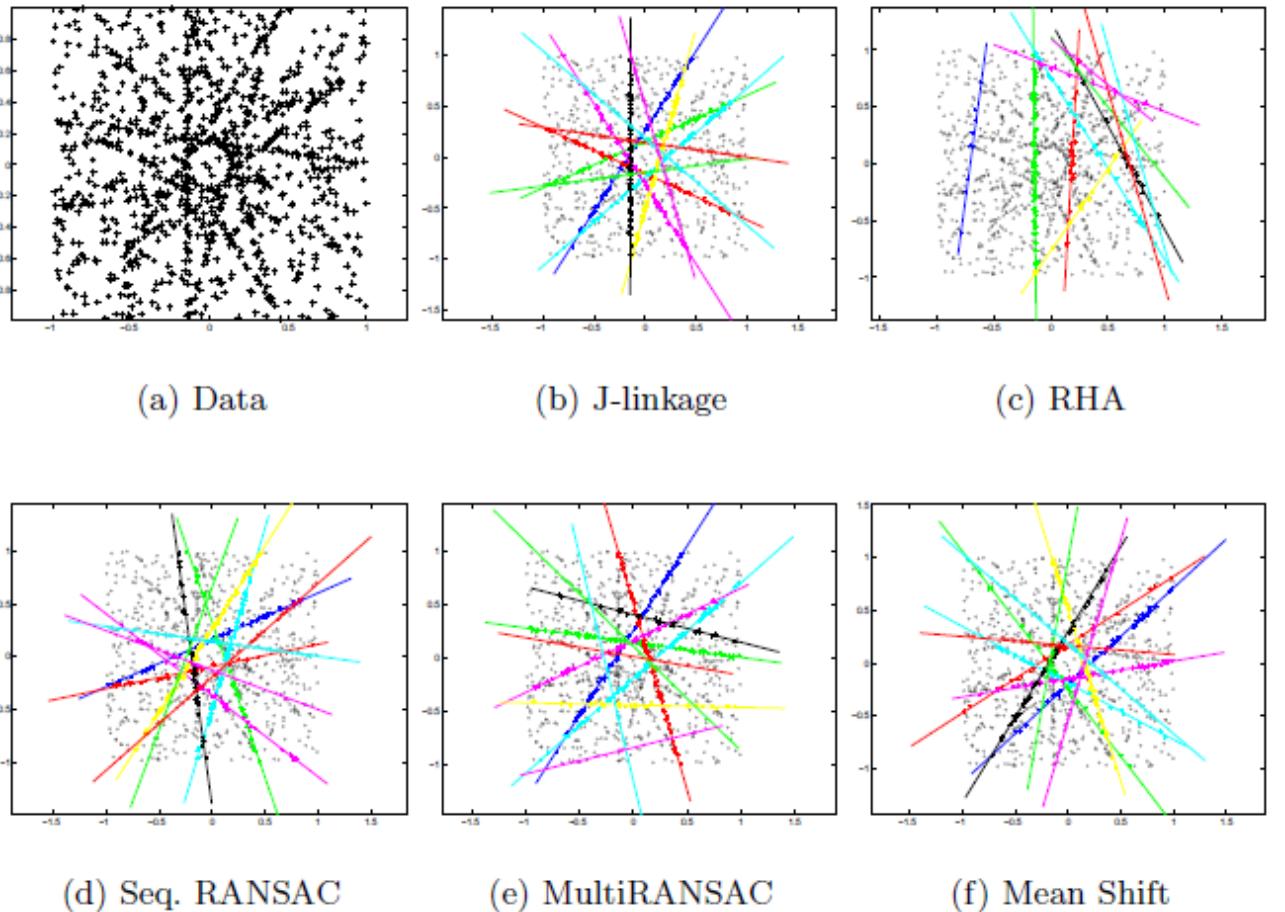


Рисунок 11 — Сравнение методов кластеризации по результатам работы на изображении одиннадцатиконечной звезды с параметрами гауссова зашумления ( $\sigma = 0.0075$ ) и числом точек шума, равным половине от общего числа. Рисунок заимствован из [22].

### 2.1.2 Однородные координаты

В проективной геометрии точки представлены в однородных координатах. Например, рассмотрим точку в двухмерном евклидовом пространстве  $(x, y) \in \mathbb{R}^2$ . Чтобы представить ее на проективной плоскости, необходимо лишь добавить третью компоненту  $z$ , равную 1, т.е.  $(x, y, 1) \in \mathbb{P}^2$ .

Для любого ненулевого  $a$  считается верным равенство  $(ax, ay, a) = (x, y, 1)$ . В случае нулевого  $a$  координаты точки вырождаются в точку  $(0, 0, 0)$ , которая не включена в  $\mathbb{P}^2$ .

Рассмотрим уравнение прямой на двухмерной евклидовой плоскости:

$$ax + by + c = 0 \quad (1)$$

Точки  $(x, y)$ , и только они, удовлетворяющие данному уравнению, являются лежащими на данной прямой. Теперь заметим, что уравнение (64) можно переписать следующим образом:

$$ax + by + c * 1 = aX + bY + cZ = 0 \quad (2)$$

Таким образом мы определили все точки  $(x, y, 1) = (X / Z, Y / Z, 1) = (X, Y, Z)$  проективного пространства  $\mathbb{P}^2$ , лежащие на прямой. Заметим также, что  $aX + bY + cZ = l^T * p = 0$ , где  $l = (a, b, c)$ ,  $p = (X, Y, Z)$ , а операция умножения  $*$  — скалярное произведение векторов.

Во-первых, важным свойством проективной геометрии является то, что уравнение линии задается вектором той же размерности, что и точки. Во-вторых, имеет место весьма красивое и простое выражение, связывающее точки и проходящие через них линии.

Не вдаваясь в подробности, запишем это и другие важные уравнения проективной геометрии, которые так или иначе понадобятся нам в работе. Ниже приведены функции меры принадлежности точки прямой (3), вычисления уравнения прямой, проходящей через две точки (4), и вычисления координат точки пересечения двух прямых (5) соответственно:

$$incidence(l, p) = l^T * p, \quad (3)$$

$$ltp(p_1, p_2) = p_1 \times p_2, \quad (4)$$

$$intersection(l_1, l_2) = l_1 \times l_2, \quad (5)$$

где оператор  $\times$  — векторное, а  $*$  — скалярное произведение.

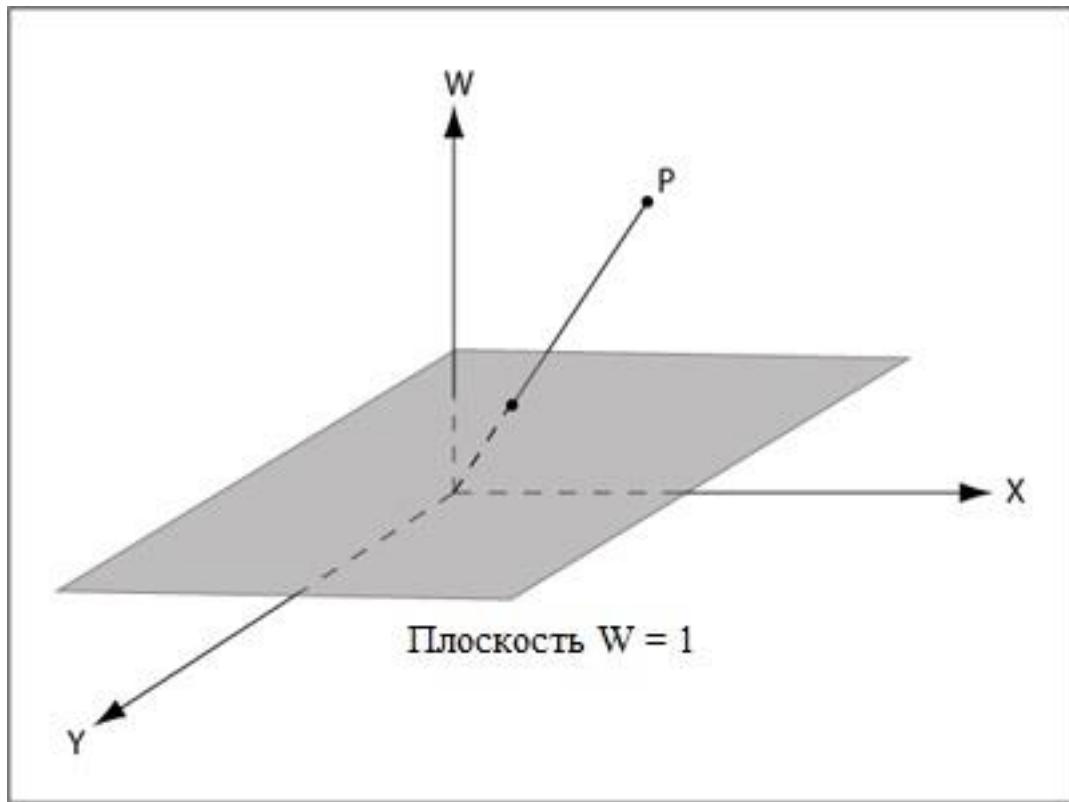


Рисунок 12 — Проекция точек из пространства  $\mathbb{R}^3$  в  $\mathbb{P}^2$ , рисунок заимствован из [13].

То есть связь между соответствующими точками пространств  $\mathbb{R}^3$  и  $\mathbb{P}^2$  можно изобразить как центральную проекцию точки  $p$  на плоскость  $W = 1$  (рис. 12).

Чтобы вернуться из  $\mathbb{P}^2$  обратно в  $\mathbb{R}^2$ , достаточно поделить координаты точки на  $z$ -координату, то есть:

$$(X, Y, Z) = (X / Z, Y / Z, 1) = (x, y, 1) \in \mathbb{P}^2 \sim (x, y) \in \mathbb{R}^2$$

Из данной процедуры сразу видно, что  $\mathbb{R}^2 \subset \mathbb{P}^2$ , так как содержит элементы с  $z = 0$ . Эти точки составляют довольно важное подмножество  $\mathbb{P}^2$  и называются идеальными. Еще их называют точками в бесконечности, так как они соответствуют предельным точкам, лежащим бесконечно далеко от начала координат. Несмотря на свой особый вид, данные точки никаким специальным образом не обрабатываются, то есть рассматриваются абсолютно также как и обычные. Все идеальные точки лежат на так называемой идеальной прямой или прямой в бесконечности.

Также заметим, что точкам  $(X, Y, Z) = (x, y, 1) \in \mathbb{P}^2$  можно поставить в соответствие прямую, проходящую через начало координат и точку  $p^*(x, y, 1) \in \mathbb{R}^3$  с выколотой точкой  $(0, 0, 0)$ . Подобным образом линия  $l(a, b, c)$  на проективной плоскости может быть визуализирована в  $\mathbb{R}^3$  плоскостью, образованной началом координат и прямой, заданной перпендикуляром  $l^*(a, b, c)$ . Тогда точкам с координатами  $(x, y, 1)$  в  $\mathbb{R}^3$  соответствует плоскость  $Z = 1$ , идеальным точкам соответствуют точки на  $Z = 0$ , а идеальной прямой — сама плоскость  $Z = 0$ .

Данная связь между  $\mathbb{R}^3$  и  $\mathbb{P}^2$  может быть легко продолжена до связи между  $\mathbb{P}^3$  и  $\mathbb{P}^2$  добавлением к точкам  $\mathbb{R}^3$  четвертой координаты 1. Такая связь очень хорошо подходит для описания преобразования проецирования трехмерной сцены на двухмерную плоскость изображения.

## **2.2 Связь между мировой системой координат и системой координат камеры**

Рассмотрим точку  $p(X, Y, Z) \in \mathbb{R}^3$ . Для того, чтобы осуществить ее перенос на вектор  $t(T_x, T_y, T_z)$ , мы можем воспользоваться следующим матричным выражением:

$$\begin{bmatrix} p + t \\ 1 \end{bmatrix} = \begin{bmatrix} I & t \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} p \\ 1 \end{bmatrix} \quad (6)$$

Похожим образом выражается поворот точки  $p$ , заданный матрицей поворота  $R$  (7), и умножение каждой из координат точки на независимый коэффициент  $d(dx, dy, dz)$  (8):

$$\begin{bmatrix} p^* \\ 1 \end{bmatrix} = \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} p \\ 1 \end{bmatrix} \quad (7)$$

$$\begin{bmatrix} dx * X \\ dy * Y \\ dz * Z \\ 1 \end{bmatrix} = \begin{bmatrix} dx & 0 & 0 & 0 \\ 0 & dy & 0 & 0 \\ 0 & 0 & dz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (8)$$

Можно заметить, что в данных выражениях точка  $p$  представлена в нормализованных координатах  $\mathbb{P}^3$  ( $X, Y, Z, 1$ ). Также очевидно, что данные выражения остаются верными и для точек вида  $\alpha p = (\alpha X, \alpha Y, \alpha Z, \alpha)$ :

$$\alpha(M * p) = M * (\alpha p)$$

Интересно рассмотреть, как данные преобразования влияют на идеальные точки  $q(X, Y, Z, 0)$ . Простой подстановкой проверяется, что:

- Перенос на вектор  $t$  оставляет идеальную точку на месте
- Поворот действует на идеальную точку абсолютно также, как и на конечную
- Масштабирование на вектор  $d(dx, dy, dz)$  действует аналогично действию на конечную точку

Теперь рассмотрим связь между системами координат камеры и мировой

системой координат. Пусть в мировой системе координаты камеры представлены точкой  $t$ , а матрица поворота  $R$  связывает соответствующие оси систем, тогда выражения связи имеет вид:

$$p_w \rightarrow p_c = R(p_w - t), \quad (9)$$

где  $p_w$  — точка в мировой системе, а  $p_c$  — в координатах системы камеры.

Расскроем скобки справа:

$$p_c = Rp_w - Rt$$

и воспользуемся нормализованными координатами:

$$\begin{bmatrix} p_c \\ 1 \end{bmatrix} = \begin{bmatrix} R & -Rt \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} p_w \\ 1 \end{bmatrix} = \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} I & -t \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} p_w \\ 1 \end{bmatrix} \quad (10)$$

Тогда матрица  $M$ , определенная как:

$$M = \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} I & -t \\ 0 & 1 \end{bmatrix}, \quad (11)$$

задает матрицу перехода между системами координат.

Матрицы  $R$  и  $t$  задают внешние (extrinsic) параметры камеры — ориентацию и позицию — в мировых координатах.

Прежде, чем перейти сформулировать задачу поиска ТСП, необходимо для начала задать используемую модель для описания камеры, а точнее преобразования, осуществляемого ей при фото- или видеосъемке.

## 2.3 Модель камеры–обскуры

Физической моделью камеры в подобных задачах обычно считают камеру–обскуру. Это светонепроницаемый ящик, в котором на одной из стенок помещен экран из белой бумаги, а на другой имеется отверстие небольшого диаметра (не более 5 мм). Такую модель иногда называют моделью булавочного отверстия. Лучи света, проходя через отверстие, проецируют перевернутое изображение объекта, находящегося вне ящика между источником света и стеной камеры с отверстием (рис. 13).

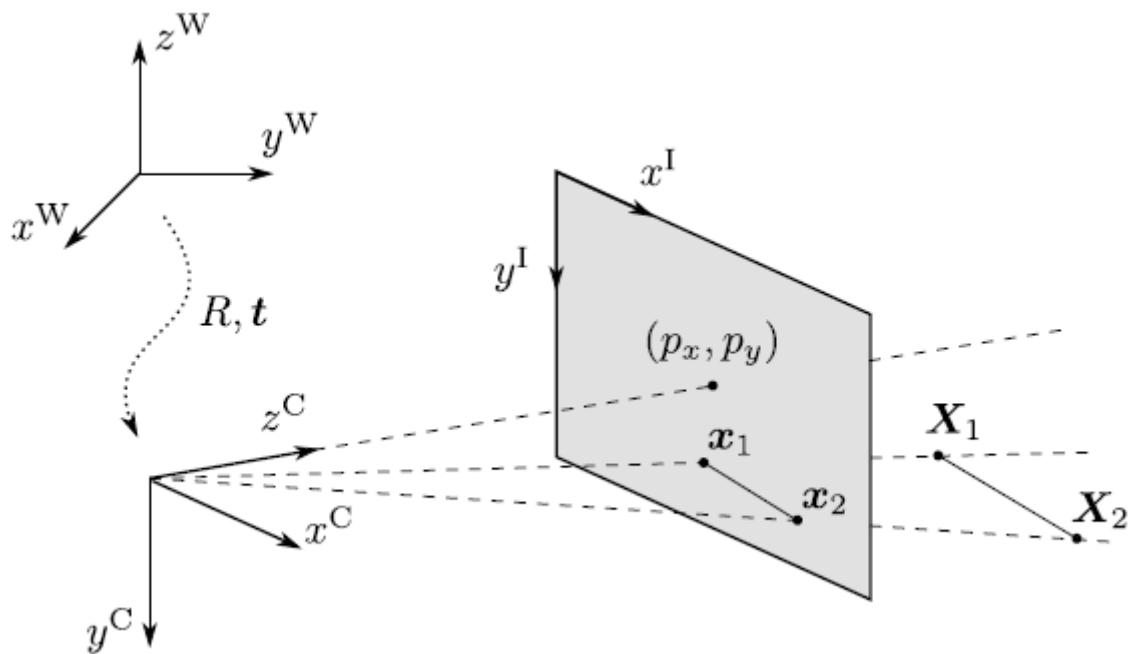


Рисунок 13 — Модель камеры–обскуры, рисунок заимствован из [1].

В качестве пояснения к рис. 13 следует отметить, что поворот камеры  $R$  и перенос  $t$  задают преобразование координат из мировой системы в систему, связанную с камерой (9). Имея известную матрицу калибровки камеры  $K$ , можно задать функцию проекции, связывающую координаты точки в мировых координатах в координаты кадра полученного изображения (14).

Таким образом, математической моделью камеры–обскуры является оператор, посредством которого точки  $p(X, Y, Z) \in \mathbb{R}^3$  проецируются на двухмерную плоскость по правилу (12):

$$(x, y) = \left( \frac{fX}{Z}, \frac{fY}{Z} \right), \quad (12)$$

или в нормализованных координатах:

$$(x, y, 1) = \left( \frac{fX}{Z}, \frac{fY}{Z}, 1 \right) = (X, Y, \frac{Z}{f}) = (fX, fY, Z)$$

Данное правило может быть записано в виде матрицы проекции, используя (8):

$$\begin{bmatrix} fX \\ fY \\ fZ \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Заметим также, что вектор  $[0 \ 0 \ 0 \ 1]^T$  является нуль–вектором нуль–пространства данного преобразования.

В практических задачах часто удобнее иметь дело с координатами, выраженными в пикселях нежели, например, в миллиметрах. Перевод координат требует информации о линейных размерах пикселя (например, в миллиметрах) и координат главной точки (principal point), которой соответствует центр изображения, то есть точка пересечения оптической оси камеры с плоскостью изображения. В общем случае главная точка может не совсем точно совпадать с центром матрицы камеры, и, более того, довольно часто центр координат в пикселях определяется одним из углов изображения.

За такого рода преобразование координат отвечает так называемая матрица калибровки камеры  $K$ , которая задает внутренние (intrinsic) параметры камеры и предполагается неизменяемой во времени:

$$K = \begin{bmatrix} \frac{f}{m_x} & s & p_x \\ 0 & \frac{f}{m_y} & p_y \\ 0 & 0 & 1 \end{bmatrix}, \quad (13)$$

где используются следующие обозначения:

- $f$  — фокусное расстояние камеры в некоторой единице длины (обычно в мм или дюймах),
- $m_x, m_y$  — линейные размеры пикселя, выраженные в той же единице длины, что и  $f$ . Таким образом  $\frac{f}{m_x}$  и  $\frac{f}{m_y}$  имеют размерность пикселей
- $s$  — коэффициент асимметрии камеры
- $p_x, p_y$  — координаты главной точки в пикселях.

Большинство цифровых камер на основе приборов с зарядовой связью (ПЗС) имеют квадратные пиксели ( $m_x = m_y$ ), нулевую асимметрию ( $s = 0$ ) и главную точку, расположенную близко к центру изображения.

Теперь мы готовы выразить проективное преобразование, отвечающее отображению трехмерной сцены на двухмерную плоскость изображения:

$$P: \mathbb{P}^3 \rightarrow \mathbb{P}^2, \quad P(p) = p^* = KR[I] - t] p \quad (14)$$

Рассмотрим прямую в  $\mathbb{P}^3$ , заданную как  $X(\alpha) = A + \alpha D$ , где  $A$  — некоторая точка на этой прямой,  $D(d, 0)$  — направляющий вектор и  $d \in \mathbb{R}^3$ ,  $\alpha \in \mathbb{R}$ . Тогда, воспользовавшись (14), найдем проекцию этой прямой:

$$\begin{aligned}
 x(\alpha) &= P(X(\alpha)) = P(A) + \alpha P(D) = P(A) + \alpha KR[I| - t]D \\
 &= P(A) + \alpha KRd,
 \end{aligned} \tag{15}$$

так как  $[I| - t]D = d$ .

Точка схождения перспективы  $vp \in \mathbb{P}^2$ , соответствующая направлению  $d$ , является предельной точкой для проекции линии  $x(\alpha)$  при  $\alpha \rightarrow \infty$  (рис. 14):

$$vp = \lim_{\alpha \rightarrow \infty} x(\alpha) = \lim_{\alpha \rightarrow \infty} P(A) + \alpha KRd = KRd \tag{16}$$

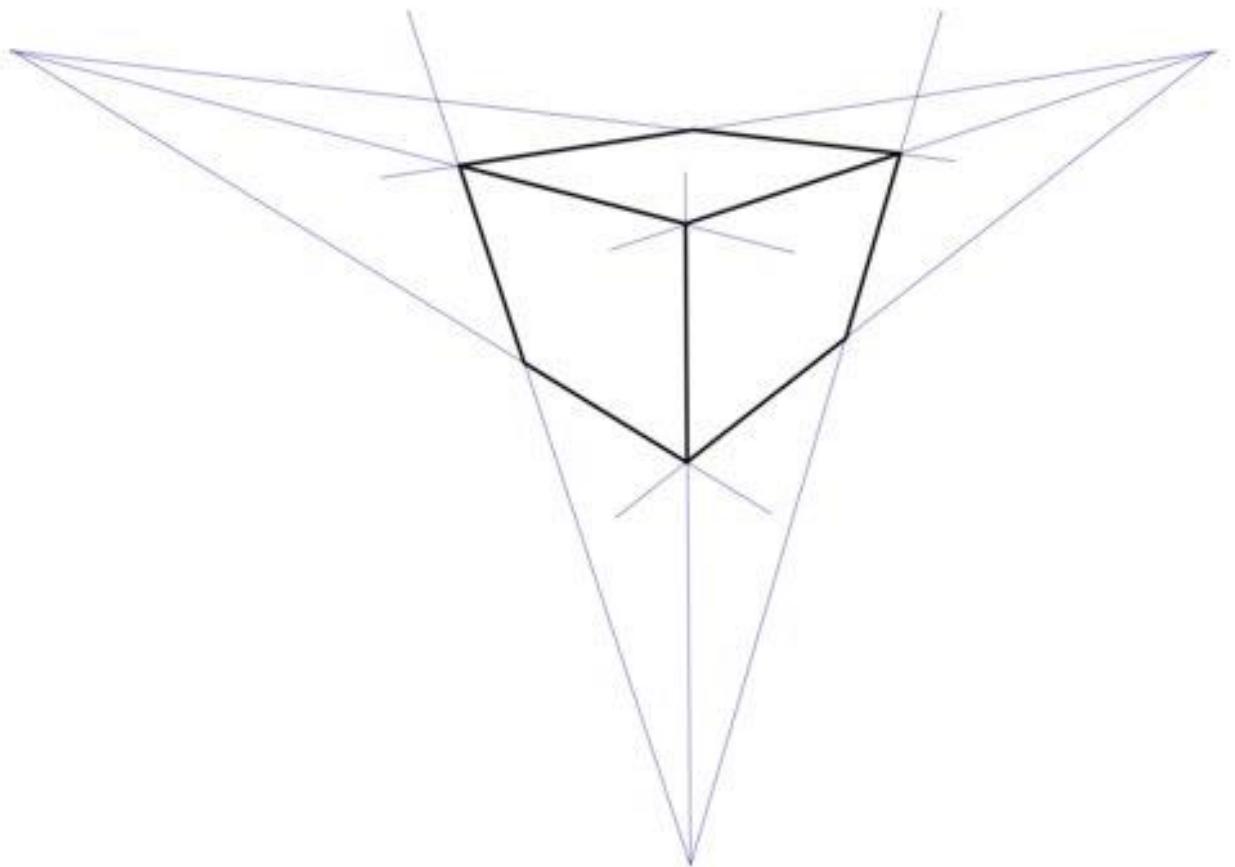


Рисунок 14 — Пример перспективного изображения куба. В данном случае ТСП, образованные направлениями ребер куба, являются конечными.

В системе координат, связанной с камерой,  $R = I$ , поэтому:

$$vp = Kd \quad (17)$$

Из полученного результата можно заключить следующее:

- ТСП  $vp$  не зависит от положения  $t$  камеры
- существует взаимно однозначное отношение между  $vp$  и вектором направления  $d$  прямой в трехмерном пространстве

Будем называть две ТСП ортогональными, если ортогональны векторы  $d$  направления соответствующих им прямых. ТСП, которые являются идеальными, называются бесконечными, иначе — конечными (рис. 14).

Большинство методов, основанных на обнаружении ТСП, работают в предположении, что на изображениях можно выделить некоторый набор сегментов линий, соответствующих взаимно ортогональным направлениям, т.е. имеющих ортогональные ТСП. Именно поэтому данные методы способны показывать хорошие результаты на изображениях помещений и городских пейзажах — объекты окружающего нас мира довольно часто имеют правильные геометрические формы и расставлены параллельно или перпендикулярно друг другу. Например, столы, полки, окна, пол, стены и потолок — внутри помещений; здания, дороги и разметка на них, окна или витрины зданий, линия крыши — вне помещений.

## **2.4 Выделение сегментов линий на изображении методом Джииои**

Первым этапом является выделение сегментов линий на изображении. Для решения было предложено использовать метод Джииои [8]. Как уже было

озвучено в 2.1.1 , это довольно новый алгоритм, отличающийся высокой скоростью по сравнению с другими алгоритмами, основанными на анализе связанных компонент градиента изображения. Вычислительная сложность алгоритма является линейной по отношению к количеству пикселей изображения. По быстродействию он уступает алгоритмам, основанным на преобразовании Хафа (Hough), но позволяет достичь более высокого качества.

Целью работы детектора линий является выделение локально строгих контуров на изображении, которые называются сегментами линий (СЛ). Контур — часть изображения, где уровень градиента серого изменяется достаточно быстро от темного к светлому и наоборот (рис. 15).

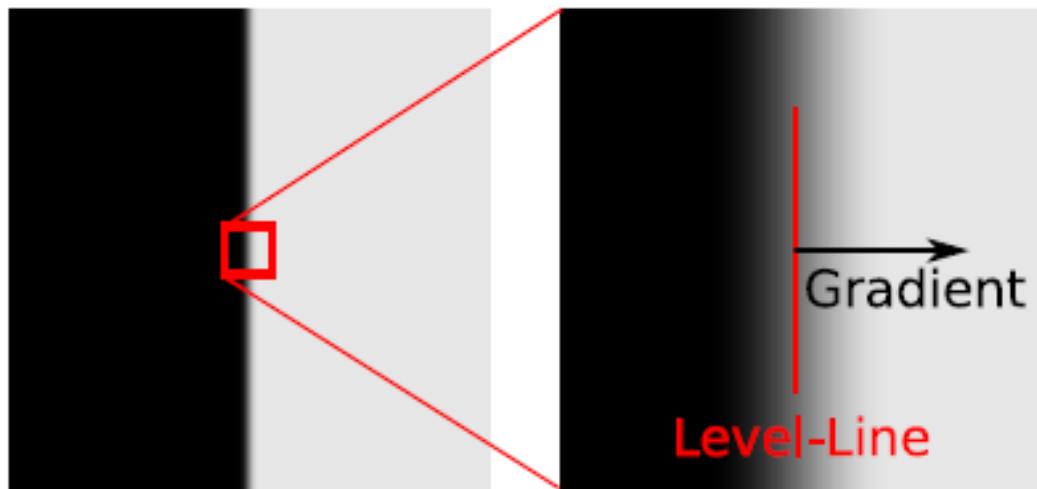


Рисунок 15 — Градиент изображения и линия уровня, рисунок заимствован из [8]

Данный алгоритм для начала строит поле линий уровня в каждом пикселе, где каждая линия уровня задает направление, перпендикулярное направлению изменения градиента серого. Затем это поле разделяется на связанные регионы линий уровня, имеющих примерно одинаковое направление. Эти связанные регионы называются зонами поддержки линии (рис. 16).

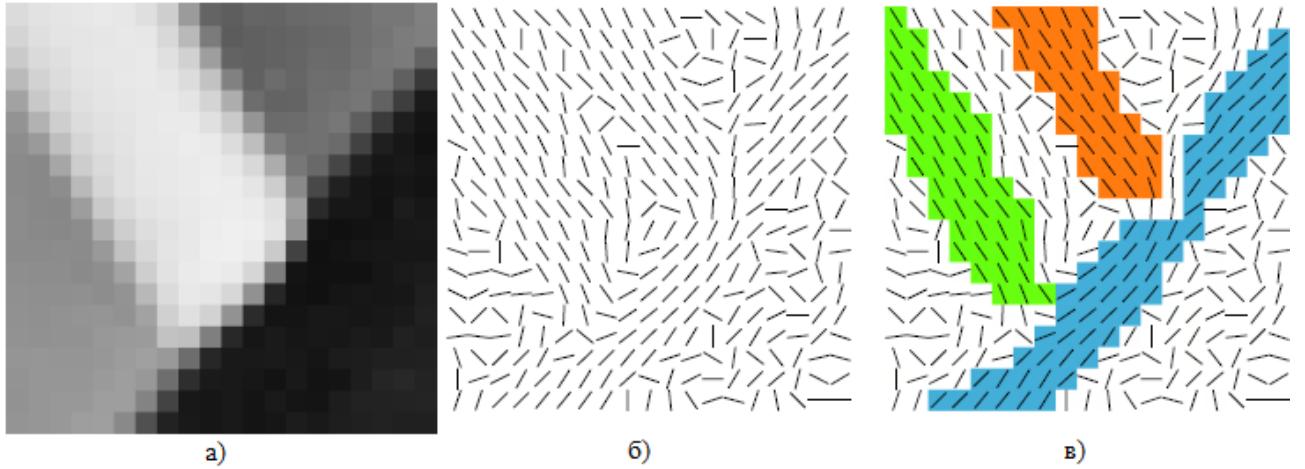


Рисунок 16 — Зоны поддержки линий: а) изображение в градиентах серого; б) поле линий уровня; в) зоны поддержки линий, выделенные цветом. Рисунок заимствован из [8].

Каждая найденная зона поддержки (ЗПЛ) линии является кандидатом на СЛ. Далее каждой ЗПЛ ставится в соответствие минимальный прямоугольник, покрывающий ее. Данный прямоугольник имеет вектор направления, совпадающий с направлением пары его длинных сторон. Соответственно можно оценить количество сонаправленных пикселей  $k$  (в примере на рис. 17 данного прямоугольника  $k = 8$  — количество сонаправленных точек), в которых направление линии уровня совпадает с направлением прямоугольника с учетом некоторого угла толерантности  $\tau$ .

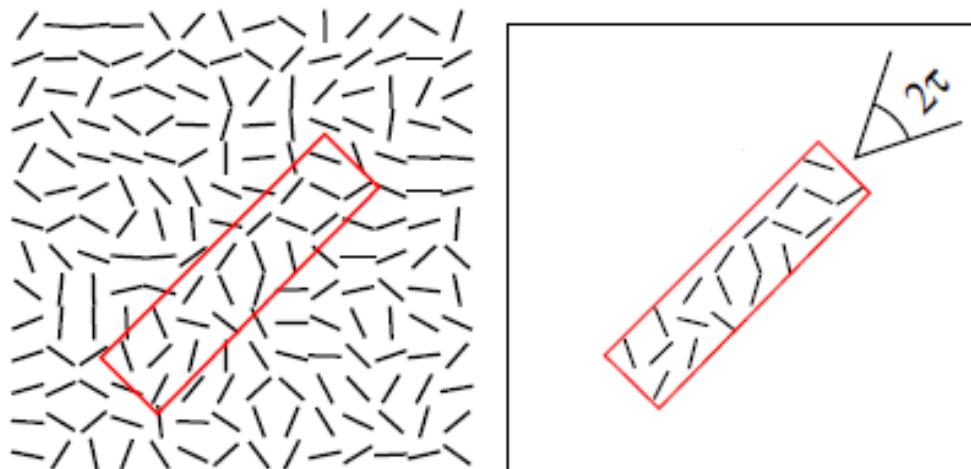


Рисунок 17 — Пример прямоугольника, соответствующего некоторой зоны поддержки линии, рисунок заимствован из [8].

противного. Для этого сравнением ведется проверка нулевой гипотезы  $H_0$  — отсутствия валидируемой структуры на изображении. Соответственно, зона поддержки линии отвергается, если она имеет недостаточно низкую вероятность в модели гипотезы  $H_0$ . Показателем выступает количество сонаправленных пикселей  $k$ .

Для этого вычисляется вероятность события  $P_{H_0}$  того, что прямоугольник  $r$  в модели  $H_0$  имеет не меньше сонаправленных точек, чем в наблюдаемой. Пусть  $k(r, i)$  — количество сонаправленных точек для прямоугольника  $r$  на изображении  $i$ , а  $n(r)$  — общее количество пикселей, в прямоугольнике. Тогда можно рассматривать следующее число событий:

$$N_{test} * P_{H_0}[k(r, I) \geq k(r, i)], \quad (18)$$

где  $N_{test}$  — количество тестов или общее количество возможных прямоугольников на рассмотрении,  $I$  — случайное изображение, удовлетворяющее модели  $H_0$ . Модель нулевой гипотезы фиксирует количество сонаправленных точек  $k(r, I)$ , которое зависит от распределения поля линий уровня изображения  $I$ . То есть  $H_0$  является моделью шума для ориентаций градиента изображения, нежели моделью шума самого изображения.

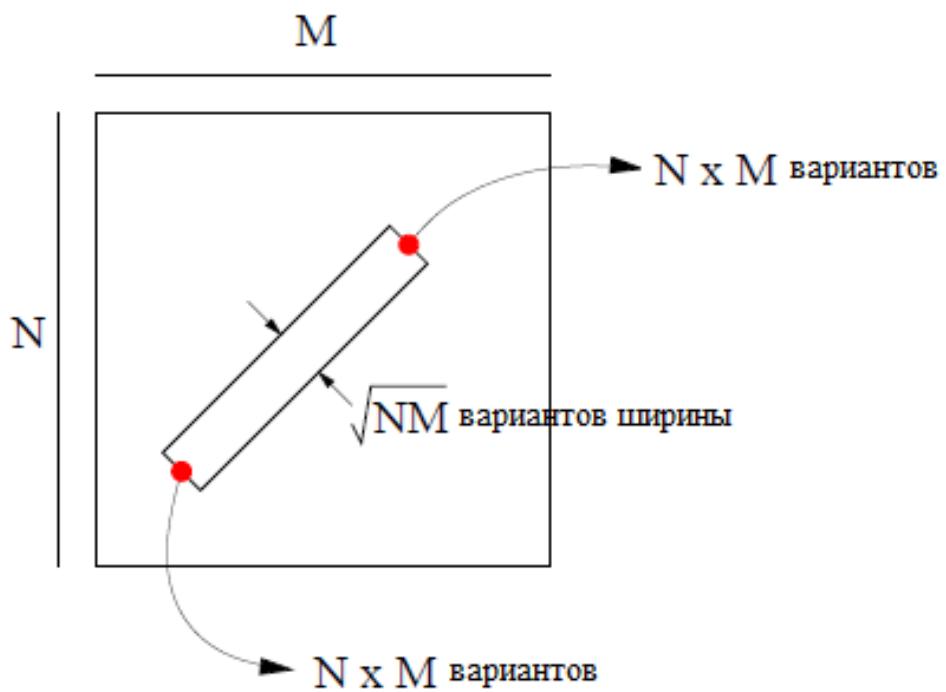


Рисунок 18 — Оценка количества количества тестов  $N_{test}$ . Рисунок заимствован из [19].

Вероятность  $P_{H_0}$  равна:

$$P_{H_0}[k(r, I) \geq k(r, i)] = B(n(r), k(r, i), p) \quad (19)$$

где  $B(n, k, p)$  — хвост биномиального распределения:

$$B(n, k, p) = \sum_{j=k}^n \binom{n}{j} p^j (1-p)^{n-j} \quad (20)$$



Рисунок 19 — Пример работы метода LSD на изображении размером  $512 \times 512$  пикселей, рисунок заимствован из [8].

Количество  $N_{test}$  (рис. 18) прямоугольников на изображении из  $N \times M$  пикселей вычисляется как количество вариантов задания начальной и конечной точки (начальная и конечная точка — центры коротких сторон) прямоугольника, умноженное на количество вариантов задания ширины, с учетом точности

$$\gamma = \frac{\tau}{\pi}:$$

$$N_{test} = (NM)^{5/2} * \gamma \quad (21)$$

Подставив значения в (18), получим число ложных срабатываний гипотезы  $H_0$ , т.е. оценку величины ошибки первого рода, для прямоугольника  $r$  на изображении  $i$ :

$$NFA(r, i) = (NM)^{5/2} * \gamma * B(n(r), k(r, i), p) \quad (22)$$

Таким образом, когда величина  $NFA(r, i)$  не превышает некоторого порога  $\epsilon$ , прямоугольник  $r$  некоторой потенциальной СЛ называется  $\epsilon$ -значимым и принимается (рис. 20–21).

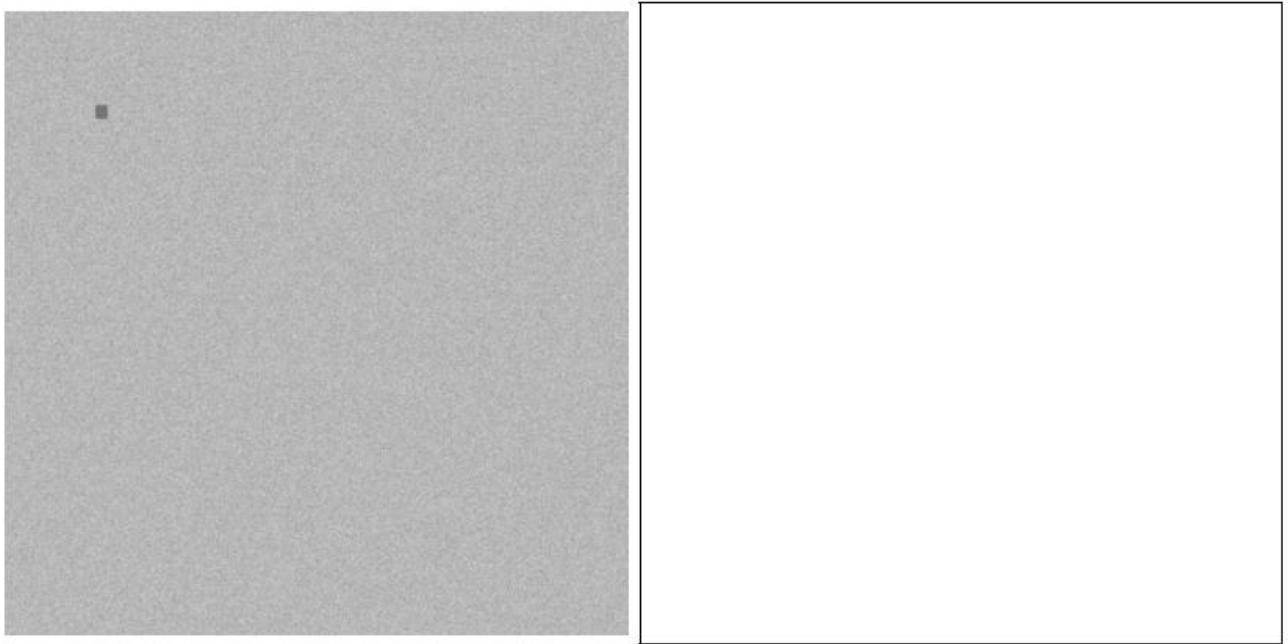


Рисунок 20 — Пример результата работы метода на маленького квадрата на фоне шума, рисунок заимствован из [8].

В примере на рис. 20 метод работает в соответствии с нашим восприятием — объект слишком мал, чтобы ему придали значение. Размер изображения  $417 \times 417$  пикселей.

На рис. 21, согласно принципу Гельмгольца, квадрат уже является хорошо воспринимаемым объектом и должен быть распознан, что и показано на правой половине рисунка.

Среднее число  $\epsilon$ -значимых прямоугольников на изображении шума не превышает  $\epsilon$ , что позволяет контролировать количество ложноположительных срабатываний алгоритма на изображении шума [8]. Это также является свидетельством того, что данный метод удовлетворяет принципу Гельмгольца. Авторами метода предлагается использовать  $\epsilon = 1$ .

Метод спроектирован таким образом, чтобы являться как можно более

автоматическим и автономным средством выделения СЛ на изображениях, не требуя кропотливых настроек входных параметров со стороны пользователя.

Основные параметры алгоритма были осторожно настроены авторами метода, чтобы подходить для подавляющего большинства изображений. Как следствие, эти параметры позиционируются как часть самого алгоритма и не предоставлены на выбор пользователю (либо выбор, отличный от значений по умолчанию, нежелателен), что делает метод очень простым и удобным в использовании, при наличии готовой реализации (рис. 19).



Рисунок 21 — Пример результата работы метода LSD на незначительном участке изображения рис. 20. Рисунок заимствован из [8].

## 2.5 Кластеризация сегментов линий методом Seq. RANSAC

Следующим этапом проводится кластеризация выделенных сегментов линий с использованием последовательного применения адаптивного алгоритма RANSAC (RANdom SAMpling Consensus) [9]. Такая вариация метода называется Sequential RANSAC.

На каждом прогоне алгоритма вычисляется самый большой из оставшихся кластер сегментов, линии которого пересекаются в одной точке с некоторой допустимой погрешностью. Полученный кластер соответствует некоторой ТСП. Всего производится 3 запуска алгоритма.

Алгоритм работает в предположении:

- Параметры могут быть оценены по  $n$  элементам. В нашем случае параметрами являются СЛ и  $n = 2$

- Всего имеется  $m$  элементов
- Вероятность случайной выборки элемента, являющегося частью искомого кластера равна  $p$
- Вероятность завершения алгоритма без нахождения хорошей оценки искомого кластера равна  $p_{fail}$

В таком случае алгоритм RANSAC состоит из последовательности итераций, состоящих из следующих шагов:

1. Случайным образом выбирается пара сегментов линий
2. Вычисляется точка их пересечения  $vr$ , которая объявляется оценкой потенциальной ТСП
3. Проводится проверка каждого сегмента линии из  $m$  имеющихся на предмет принадлежности его к кластеру потенциальной ТСП с заданными параметрами допустимости точности. Обозначим отношение мощности кластера к мощности всего множества имеющихся СЛ буквой  $r$ .
4. Если  $r$  достаточно велико, полученный кластер принимается и алгоритм завершает свою работу
5. Шаги 1..4 повторяются  $k$  раз
6. Алгоритм завершается с ошибкой

Рассмотрим немного подробнее шаги алгоритма в терминах решаемой задачи. Оценка ТСП  $vr$  вычисляется в однородных координатах на основе случайно выбранных двух СЛ по формуле (5).

Для проверки принадлежности сегмента  $l$  кластеру некоторой потенциальной ТСП  $vr$  задается функция расстояния. Она определяется как угол  $\alpha$  между линией, заданной сегментом, и линией, связывающей точку  $vr$  и центр сегмента  $l_m$ :

$$dist(vp, l) = \left| \arcsin \left( \frac{incidence(ltp(vp, l_m), l_A)}{\|ltp(vp, l_m)\|_{xy} * \|l_m - l_A\|_{xy}} \right) \right|, \quad (23)$$

где  $l_A$  — точка начала сегмента  $l$ ,  $\|p\|_{xy} = \sqrt{p_x^2 + p_y^2}$  — норма, учитывающая только первые две координаты, а функции  $incidence(l, p)$  и  $ltp(p_1, p_2)$  вычисляются по формулам (3) и (4) соответственно.

Расстояние между ТСП  $v$  и сегментом  $l$  (рис. 22) вычисляется как абсолютная величина угла между сегментом  $l$  и прямой, соединяющей точку  $v$  и центр сегмента  $l_m$ .

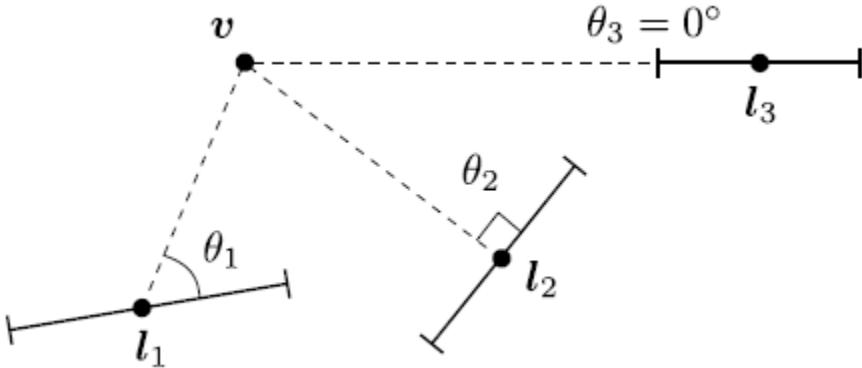


Рисунок 22 — Углы между ТСП  $v$  и сегментом  $l$ , рисунок заимствован из [1].

Тогда функция–индикатор принадлежности СЛ  $l$  кластеру ТСП  $vp$  с допустимой погрешностью  $\epsilon_\alpha$  задается следующим образом:

$$\delta(vp, l, \epsilon_\alpha) = dist(vp, l) < \epsilon_\alpha \quad (24)$$

По заданной функции–индикатору определяется кластер СЛ для имеющейся потенциальной ТСП. Назовем содержащиеся в нем сегменты внутренними, а все остальные — внешними по отношению к данной  $vp$ . Тогда число  $r$  — отношение количества внутренних сегментов к числу всех сегментов

*m*. Чем выше число  $r$ , тем более подходящей считается ТСП.

Однако заранее сказать, какое значение  $r$  является достаточным для выполнения условия на шаге 4, невозможно. Поэтому на каждой итерации производится адаптация оценки достаточного количества итераций  $k$  для имеющегося значения  $r$  наилучшего найденного кластера.

Число итераций алгоритма определяется следующим образом. Представим, что мы ищем некоторую подходящую нам ТСП. Вероятность выбора внутреннего сегмента равна  $r$ , а вероятность события, что во всем множестве сегментов мы случайно выберем оба внутренних сегмента —  $r^2$ . Тогда вероятность выбора хотя бы одной внешней —  $1 - r^2$ . Соответственно, вероятность события, когда за  $k$  итераций ни разу не будет выбрана пара внутренних сегментов равна:

$$P(k) = (1 - r^2)^k \quad (25)$$

Заметим, что  $P(k)$  — строго убывающая функция. Теперь нам хотелось бы гарантировать с вероятностью  $p$ , что за некоторое количество итераций  $k$  будет выбрана хотя бы одна пара внутренних сегментов линий:

$$\begin{aligned} p &\geq 1 - P(k) \\ \Rightarrow P(k) &\geq 1 - p \\ \Rightarrow (1 - r^2)^k &\geq 1 - p \end{aligned}$$

Логарифмируя обе стороны, получаем:

$$k \geq \frac{\log(1 - p)}{\log(1 - r^2)} \quad (26)$$

Таким образом на каждой итерации производится переоценка параметра  $k$  по формуле (64) аппроксимацией снизу значения  $r$  величиной  $r'$ , соответствующей наилучшей из ТСП, найденных за текущие итерации.

После каждого запуска алгоритма RANSAC внутренние сегменты для найденной ТСП удаляются из выборки и в последующих запусках не участвуют. В итоге после трех последовательных запусков алгоритма мы имеем три кластера сегментов линий, каждому из которых соответствует довольно грубая оценка ТСП.

## 2.6 Уточнение ТСП

На данном этапе имеются найденные кластеры сегментов линий с приближенными оценками ТСП.

Рассмотрим сегменты одного из кластера. Линия  $l$ , заданная каждым из сегментов, содержит точку  $vp$ , являющуюся оценкой ТСП данного кластера:

$$incidence(l, vp) = l^T * vp = 0 \quad (27)$$

Для всего множества сегментов кластера получается следующая СЛАУ:

$$A_l^T * vp = [l_1 \dots l_n]^T * vp = 0 \quad (28)$$

Так как включение СЛ в кластер имеет приближенный характер с заданным уровнем допуска, как и сами координаты СЛ, координаты точки схождения перспективы, выбранной на основе случайной пары из этого множества, не учитывает «вклад» остальных сегментов кластера и являются грубым приближением. Логично, что для полученного приближения уравнение

(64) в реальных условиях скорее всего не выполняется. Более того, с большой вероятностью данная СЛАУ не имеет точного решения.

Таким образом имеется переопределенная система уравнений. Требуется найти такую нетривиальную точку  $vp$ , которая минимизирует функционал в левой части, а именно:

$$vp = \underset{x}{\operatorname{argmin}}(A_l^T * x) \quad (29)$$

Прежде, чем перейти к решению данной задачи, произведем некоторое преобразование задачи. Для этого отметим, что до сих пор мы работали в координатах конечного изображения, выраженного в пикселях. В работе Чиполлы [14] предлагается преобразовать их в нормированные координаты изображения, соответствующие камере с фокусным расстоянием равным 1 и главной точкой, расположенной в начале координат и выраженные в реальных единицах длины (например метры, дюймы, миллиметры).

Чтобы перейти к нормированным координатам, домножим слева координаты концов сегментов линий на матрицу, обратную матрице калибровки камеры  $K$ , и заново вычислим уравнения прямых, соответствующих данным сегментам:

$$L = lcp(K^{-1}l_A, K^{-1}l_B) \quad (30)$$

По полученным уравнениям прямых в нормированных координатах строится СЛАУ, аналогичная (28):

$${A_L}^T * d_{vp} = [L_1 \dots L_n]^T * d_{vp} = 0 \quad (31)$$

И задача минимизации (29) превращается в:

$$d_{vp} = \underset{x}{\operatorname{argmin}}({A_L}^T * x) \quad (32)$$

Существуют разные методы решения данной задачи оптимизации. В данной работе был выбран метод, использующий сингулярное разложение матрицы  $A_L^T$ . Тогда сингулярный вектор, соответствующий наименьшему ненулевому сингулярному значению матрицы  $A_L^T$ , минимизирует функционал слева из (64).

Полученная точка  $d_{vp}$  задает направление ТСП  $vp$  изображения. Далее производится нормировка их координат, т.е. вычисление единичных векторов направлений ТСП.

### **2.6.1 Уточнение матрицы поворота камеры относительно мировой системы координат**

В случае, если все три найденные направления ТСП взаимно ортогональны, каждый из них является вектором, в который переходит одна из осей мировой системы координат при переходе из системы координат камеры в мировую. На данном этапе существует несколько проблем.

Если количество найденных направлений меньше двух, то определить углы ориентации камеры не представляется возможным. В этом случае метод не дает ответа на поставленную задачу.

```
roll = 0; pitch = 0; yaw = 0;
```



Рисунок 23 — Пример результата работы приложения на синтетическом тесте изображения прямоугольника.

В случае, если количество найденных направлений равно двум, третье направление может быть вычислено как векторное произведение найденных двух направлений.

На рис. 23 приведен пример синтетического теста, на котором третье направление успешно вычисляется на основе найденных двух. Всего выделено только два кластера сегментов линий, по которым восстановлено направление третьей ТСП (красного цвета). Для оценки месторасположения центра, изображение обрамлено рамкой. ТСП, соответствующие координатам  $x$  и  $y$ , на изображении находятся в точках  $(1, 0)$  и  $(0, 1)$  соответственно и являются идеальными в однородных координатах.

Тройка направлений может быть не взаимно ортогональной. В этом случае производится ортогонализация матрицы  $D = [d_1 \ d_2 \ d_3]$ , составленной из единичных векторов направлений найденных ТСП. Ближайшую в смысле нормы Фробениуса ортогональную матрицу  $\widehat{D}$  можно вычислить, воспользовавшись сингулярным разложением матрицы  $D$ . Тогда:

$$\widehat{D} = UV^T, \quad (33)$$

где  $U$  и  $V^T$  определяются из сингулярного разложения матрицы  $D = U\Sigma V^T$ .

Теперь, чтобы составить матрицу поворота  $R$  системы координат камеры относительно мировой системы координат, необходимо определить к каким осям относятся какие из полученных векторов. Так как направления вычислены с точностью до знака, то необходимо определить и его. Обе задачи можно свести к нахождению матрицы поворота, минимизирующей сумму квадратов углов поворота и сохраняющей знак детерминанта. На практике, мы ее решаем следующим образом:

- Для координат  $x$  и  $y$  поочередно среди трех векторов выбирается тот, чье абсолютное значение координаты максимально.
- Выбранный вектор относится той оси, по чьей координате велся поиск.
- Затем полученный вектор умножается на  $-1$ , если соответствующая координата отрицательна
- Соответствие вектора координате  $z$  определяется по остаточному принципу. Для него лишь достаточно поправить знак направления.

В итоге, переставив столбцы матрицы  $\widehat{D}$  и поправив знаки, мы можем восстановить матрицу поворота  $R$ :

$$R = \widehat{D}^T \quad (34)$$

## 2.6.2 Определение ориентации по ТСП

Для определения углов ориентации камеры по имеющейся матрице поворота  $R$  для начала необходимо определиться с самой нотацией задания углов. В различных областях науки, в которых возникает необходимость задания углов ориентации тела, используются разные нотации.

Среди наиболее распространенных — углы Эйлера — углы поворота вокруг осей  $x$ ,  $y$  и  $z$ . Пусть углы поворота относительно некоторого набора осей равны  $\phi$ ,  $\theta$ ,  $\psi$ . Будем называть вектором углов Эйлера следующий вектор:

$$u = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \quad (35)$$

Однако и в этом случае существует 12 различных нотаций, отличающихся последовательностью применения поворотов относительно выбранных осей. В данной работе было решено использовать углы Эйлера в нотации (1, 2, 3), которой соответствует последовательно поворотов относительно осей  $z$ ,  $y$  и  $x$  соответственно:

$$R(u) = R_1(\phi) * R_2(\theta) * R_3(\psi) \quad (36)$$

Числа в нотации (1, 2, 3) как раз обозначают индексы осей в произведении матриц поворотов формулы (36). Стоит помнить, что умножение матриц производится справа налево, поэтому первый поворот осуществляется матрицей  $R_3(\psi)$ . Обозначенным углам поворота даны названия: крен, тангаж и рыскание (roll, pitch, yaw). Эти названия берут свое начало из морской навигации.

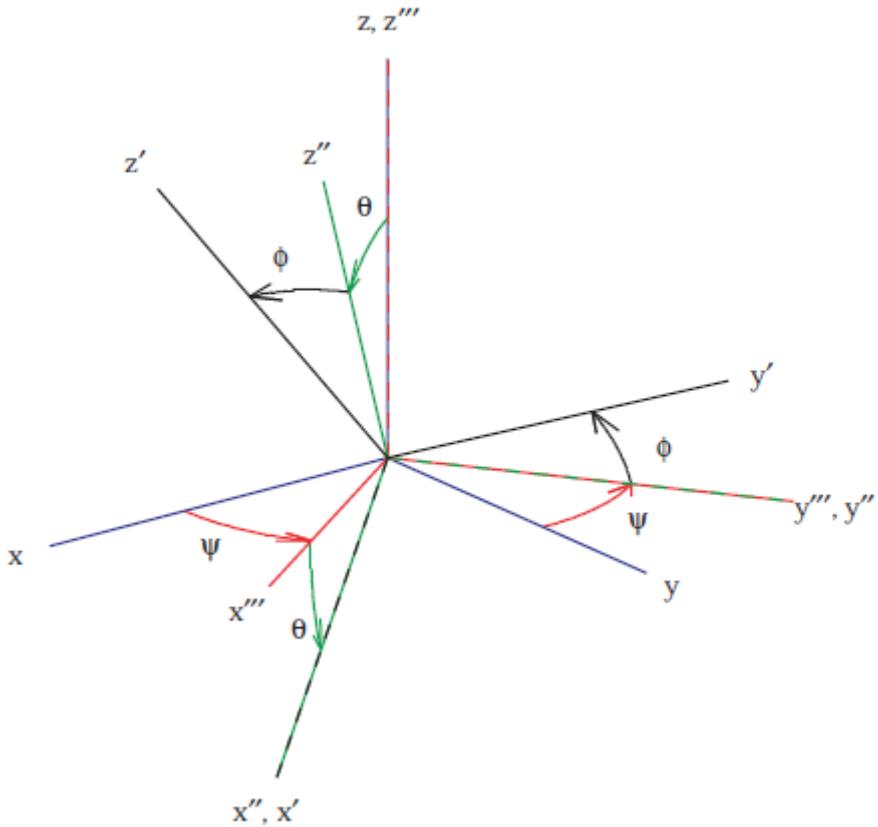


Рисунок 24 — Углы Эйлера в нотации (1, 2, 3).

В примере на рис. 24 сначала производится поворот относительно оси  $z$  на угол  $\psi$ , затем вокруг оси  $y$  на угол  $\theta$ , и последним осуществляется поворот вокруг оси  $x$  на угол  $\phi$ . Данная последовательность поворотов переводит оси координат  $(x, y, z)$  в  $(x', y', z')$ .

По имеющейся матрице поворота  $R$  вектор углов Эйлера вычисляется так [16]:

$$u_{123}(R) = \begin{bmatrix} \phi_{123}(R) \\ \theta_{123}(R) \\ \psi_{123}(R) \end{bmatrix} = \begin{bmatrix} \text{atan2}(r_{23}, r_{33}) \\ -\text{asin}(r_{13}) \\ \text{atan2}(r_{12}, r_{11}) \end{bmatrix}, \quad (37)$$

где функция  $\text{asin}$  — реализация функции  $\arcsin$ , а двухаргументная функция  $\text{atan2}$  задается через реализацию функции  $\arctangent$  —  $\text{atan}$  — следующим

образом:

$$atan2(y, x) = \begin{cases} \text{atan}\left(\frac{y}{x}\right), & x > 0 \\ \text{atan}\left(\frac{y}{x}\right) - \pi, & x < 0, y < 0 \\ \text{atan}\left(\frac{y}{x}\right) + \pi, & x < 0, y > 0 \\ \frac{\pi}{2}, & x = 0, y > 0 \\ -\frac{\pi}{2}, & x = 0, y < 0 \\ \text{неопределено,} & x = 0, y = 0 \end{cases} \quad (38)$$

## ГЛАВА 3. ПРОЕКТИРОВАНИЕ ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В рамках работы разработано тестовое приложение, которое состоит из следующих функциональных блоков:

- организация конфигурации запуска приложения и управления тестированием
- управление процессом обработки изображения
- управление процессом тестирования набора изображений

В разделах ниже описаны структуры данных приложения. Все они объединены в единое пространство имен `gutosam`. Для краткости оно опущено в именах структур и глобальных функций. Пространства имен подключаемых библиотек наоборот сохранены для ясности.

### **3.1 Блок организации запуска приложения и управления тестированием**

На вход приложение получает путь к файлу изображения (или директории базы данных изображений), а также набор необязательных параметров и ключей. По ключам входных данных определяется тип и начальная конфигурация запуска. Подробное описание ключей описано в разделе руководства пользователя 4.2 . Результатом начальной конфигурации является объекта структуры `Settings`. Описание этой структуры приведено в таблице 1.

Таблица 1 — Описание полей структуры `Settings`.

| Поле              | Тип  | Описание   |
|-------------------|------|--|
| TRACE_ENABLED     | bool | Включение трассировки  |
| BUILD_IMAGE       | bool | Нужно ли строить изображение–результат   |
| DRAW_RAW_SEGMENTS | bool | Нужно ли отображать все найденные сегменты линий, прошедшие фильтрацию   |
| SHOW_IMAGE        | bool | Включение вывода окна с изображением–результатом   |
| WAIT_AFTER        | bool | Следует ли ожидать ввода пользователя с клавиатуры перед окончанием работы   |
| SCALE_ENABLED     | bool | Включение масштабирования входного изображения под стандартный размер:<br>$640 \times 480$ при <code>POCKET_SIZE = true</code><br>$1280 \times 960$ при <code>POCKET_SIZE = false</code> |

| <b>Поле</b>             | <b>Тип</b> | <b>Описание</b>  |
|-------------------------|------------|--|
| POCKET_SIZE             | bool       | При включенном масштабировании задает размер маленького экрана |
| YORK_URBAN_DB_TEST_MODE | bool       | Включение режима тестирования базы данных изображений          |

Стоит отметить, что конфигурация на основе объекта типа *Settings* является собой нечто вроде глобальной, не зависящей от входных изображений, конфигурацией. Помимо нее имеются также конфигурации, специфичные для текущего режима запуска.

Приложение может работать в одном из двух режиме:

- Режим обработки одного изображения
- Режим тестирования базы данных изображений

Процесс работы первого режима осуществляется с помощью объекта класса *ImageProcessor*. Специфичной для него конфигурацией является конфигурация типа *SingleRunConfig*, описание которой дано в таблице 2. А в таблице 3 приведено описание методов данного класса.

Таблица 2 — Описание полей типа *SingleRunConfig*.

| <b>Поле</b>           | <b>Тип</b>  | <b>Описание</b>                                       |
|-----------------------|-------------|---|
| originPath            | std::string | Входной аргумент работы приложения пути к изображению |
| calibrationMatrixPath | std::string | Входной аргумент работы приложения пути к файлу       |

| <b>Поле</b>            | <b>Тип</b>  | <b>Описание</b>   |
|------------------------|-------------|---|
|                        |             | матрицы калибровки камеры   |
| identificationPathPart | std::string | Подстрока поля originPath, на основе которой образуются строки путей входных и выходных данных, соответствующих обрабатываемому изображению |
| extension              | std::string | Строка расширения файла входного изображения  |

Таблица 3 — Описание методов класса SingleRunConfig.

| <b>Метод</b>                                | <b>Описание</b>  |
|---|--|
| std::string getInputImagePath()             | Получить строку пути входного файла изображения  |
| std::string getCalibrationMatrixPath()      | Получить строку пути файла матрицы калибровки  |
| std::string getOutputImagePath()            | Получить строку пути выходного файла изображения—результата                                      |
| std::string getOutputNonOrthogonalVpsPath() | Получить строку пути выходного файла для матрицы направлений ТСП перед процессом ортогонализации |
| std::string getOutputOrthogonalVpsPath()    | Получить строку пути выходного файла для матрицы ортогонализированных направлений ТСП            |
| std::string getOutputAnglesPath()           | Получить строку пути выходного файла для вектора углов Эйлера                                    |

Второй режим работает на основе первого и представляет собой последовательный запуск процесса обработки каждого изображения базы. Данный режим проводится под управлением объекта класса YorkUrbanDbTester. По аналогии с ImageProcessor для объектов данного класса требуется специфичная для него конфигурация. Ей соответствует тип данных YorkUrbanDbTestRunConfig, описание полей которого приведено в таблице 4, а описание методов — в таблице 5.

Таблица 4 — Описание полей класса YorkUrbanDbTestRunConfig.

| <b>Поле</b>            | <b>Тип</b>  | <b>Описание</b>   |
|------------------------|-------------|---|
| originPath             | std::string | Входной аргумент работы приложения пути к изображению   |
| calibrationMatrixPath  | std::string | Входной аргумент работы приложения пути к файлу матрицы калибровки камеры   |
| identificationPathPart | std::string | Подстрока поля originPath, на основе которой образуются строки путей входных и выходных данных, соответствующих обрабатываемому изображению |
| extension              | std::string | Строка расширения файла входного изображения  |

Таблица 5 — Описание методов класса YorkUrbanDbTestRunConfig.

| <b>Метод</b> | <b>Описание</b> |
|--------------|-----------------|
|--------------|-----------------|

| <b>Метод</b>  | <b>Описание</b>   |
|---|---|
| <code>std::string getImageNamesListPath()</code>  | Получить строку пути входного файла со списком названий изображений тестового набора                                  |
| <code>std::string getCalibrationMatrixPath()</code>   | Получить строку пути файла матрицы калибровки   |
| <code>std::string getSingleRunbasePath(<br/>const std::string &amp;imageFolder<br/>)</code>                   | Получить специфичную для изображения базу строки пути входного файла изображения на основе имени папки с изображением |
| <code>std::string getSingleRunImagePath(<br/>const std::string &amp;basePath<br/>)</code>                     | Получить строку пути входного файла для обработки   |
| <code>std::string getOutputSingleReportPath(<br/>const std::string &amp;basePath<br/>)</code>                 | Получить строку пути выходного файла отчета тестирования для одного изображения                                       |
| <code>std::string getSingleRunGroundTruthPath(<br/>const std::string &amp;basePath<br/>)</code>               | Получить строку пути входного файла с матрицей верных направлений ТСП до процесса ортогонализации                     |
| <code>std::string<br/>getSingleRunOrthogonalGroundTruthPath(<br/>const std::string &amp;basePath<br/>)</code> | Получить строку пути входного файла с матрицей верных ортогонализированных направлений ТСП                            |
| <code>std::string getOutputGlobalReportPath()</code>  | Получить строку пути выходного файла отчета проведенного тестирования   |

### 3.2 Управление процессом обработки изображения

Ранее упоминалось, что управление процессом обработки одного изображения осуществляется объектами класса `ImageProcessor`. Запуску процесса соответствует вызов метода `ImageProcessor::process`. Листинг 1 содержит тело данного метода.

*Листинг 1 — Метод `ImageProcessor::process()`*

```
1. SingleRunResult result;
2. TimeCounter timeCounter;
3. timeCounter.StartCounter();
4.
5. initColors();
6. loadImage();
7. loadCalibrationMatrix();
8.
9. extractLineSegments();
10. std::vector<cv::Point3d> vps;
11. for (int i = 0; i < 3; i++)
12. {
13.     if (notUsedSegments.size() < 2)
14.         break;
15.     cv::Point3d refinedVpNormalized = findVanishingPoint(vps);
16.     vps.push_back(refinedVpNormalized);
17. }
18.
19. result.vpBasis = getRotationMatrix(vps);
20. refineVpBasis(result.vpBasis);

21. result.orthoVpBasis = getNearestOrthogonalMatrix(result.vpBasis);
22. reorderColumn(result.orthoVpBasis, 1);
23. reorderColumn(result.orthoVpBasis, 0);
24. reorderColumn(result.orthoVpBasis);
25.
```

```

26. result.eulerAngles = getEulerAngles(result.orthoVpBasis.t());
27. result.runTime = timeCounter.GetCounter();
28. saveResults(result);
29. drawProcessedImage(getEulerAnglesString(result.eulerAngles));
30. return result;

```

Из приведенного листинга видно, что производится замер времени выполнения всего процесса обработки изображения.

В начале работы происходит чтение входного файла изображения и матрицы калибровки, инициализация внутренних полей. Описание внутренних полей приведено в таблице 6.

Таблица 6 — Описание полей класса `ImageProcessor`.

| Поле                      | Тип   | Описание  |
|---------------------------|---|---|
| config                    | <code>SingleRunConfig</code>                | Объект специфичных настроек конфигурации.   |
| settings                  | <code>Settings</code>                       | Объект глобальных настроек конфигурации.  |
| calibrationMatrix         | <code>cv::Mat</code>                        | Матрица калибровки $K$ .  |
| inversedCalibrationMatrix | <code>cv::Mat</code>                        | $K^{-1}$  |
| image                     | <code>cv::Mat</code>                        | Матрица изображения   |
| segments                  | <code>std::vector&lt;LineSegment&gt;</code> | Список всех распознанных сегментов, прошедших фильтрацию.   |
| notUsedSegments           | <code>std::set&lt;int&gt;</code>            | Множество—индикатор, которое содержит индексы сегментов линий из <code>segments</code> , участвующих в кластеризации. |

| Поле   | Тип                     | Описание                    |
|--------|-------------------------|-----------------------------|
| colors | std::vector<cv::Scalar> | Палитра используемых цветов |

Первым этапом процесса обработки является выделение сегментов линий.

### 3.2.1 Выделение сегментов линий

Для решения данной задачи в приложении использована реализация детектора сегментов линий Джиии в библиотеке OpenCV версии 3.0.0 beta [23]. Ей отвечает класс `LineSegmentDetector` (LSD), который принимает на вход изображение в градациях серого (в OpenCV ему соответствует тип `CV_8UC1`), и различные параметры настройки алгоритма. Как уже было отмечено в 2.4 , данный метод позиционируется как автоматическое средство выделения СЛ и имеет проверенный тестированием набор настроек параметров. Поэтому в качестве параметров настройки были использованы настройки, рекомендованные по умолчанию.

В листинге 2 приведен метод выделения сегментов линий с использованием класса `cv::LineSegmentDetector` и последующей их фильтрации по длине.

*Листинг 2 — Method `ImageProcessor::extractLineSegments()`*

```

1. // to grayscale
2. cv::Mat grayImage;
3. cvtColor(image, grayImage, cv::COLOR_BGR2GRAY);
4. // Detect the lines
5. std::vector<cv::Vec4i> lines_std;
6. cv::Ptr<cv::LineSegmentDetector> ls = cv::createLineSegmentDetector(
7.         cv::LSD_REFINE_STD
8. );
9. ls->detect(grayImage, lines_std);

```

```

10. segments = LineSegment::toLineSegments(
11.     lines_std,
12.     findMinAllowedLineSegmentLength(image)
13. );
14. notUsedSegments.clear();
15. for (int i = 0; i < segments.size(); i++)
16.     notUsedSegments.insert(i);

17. if (settings.BUILD_IMAGE && settings.DRAW_RAW_SEGMENTS)
18.     drawFoundSegments(segments, image, colors[0], 1);
19. if (settings.TRACE_ENABLED)
20.     std::cout << "Raw segments: " << lines_std.size() << "; passed
length filter: " << segments.size() << std::endl;

```

Результатом работы модуля LSD является список найденных сегментов линий. Для каждого сегмента дается следующая информация:

- координаты его концов в пикселях в виде четверки целых чисел
- ширина линии
- точность, с которой найден сегмент
- число ложных срабатываний (number of false alarms, NFA) (см. (22)) в области сегмента линии в виде логарифмической шкалы качества детектирования

Получив результат работы детектора сегментов линий, далее проводится фильтрация сегментов по длине. Допускаются только те, чья длина больше некоторого параметра  $min_{SegLineLength}$  зависящего линейно от длины наибольшей из сторон обрабатываемого изображения (рис. 25).



Рисунок 25 — Пример результата работы детектора сегментов линий LSD библиотеки OpenCV. Зеленым цветом отрисованы сегменты, прошедшие фильтрацию по длине, красным — отброшенные.

Было замечено, что обычно подавляющее число относительно коротких СЛ относятся к ошибочным направлениям, чем существенно зашумляют результаты работы на последующих этапах. Более того, даже в случае, когда они являются внутренними сегментами по отношению к кластерам искомых ТСП, благодаря меньшей точности направлений коротких СЛ страдает результирующая точность определения ТСП. Это связано с тем, что погрешность детектора в один пиксель при определении концов коротких сегментов ведет к значительному отклонению угла наклона всего сегмента.

Как побочное следствие, отвергая значительную часть найденных сегментов, алгоритм работает значительно быстрее, так как:

- сокращается число сегментов, которые требуется проверить на

принадлежность к кластерам на этапе кластеризации методом RANSAC

- за счет увеличения процента внутренних СЛ по отношению к общему числу сегментов, уменьшается число требуемых итераций алгоритма RANSAC
- меньше размерность переопределенной СЛАУ на этапе уточнения ТСП

Основная доля тестирования производилась на изображениях размером  $640 \times 480$  пикселей, поэтому оптимальное значение параметра фильтрации подбиралось именно под него — опытным путем было выбрано значение  $min_{SegLineLength} = 16$ . Для изображений других размеров величина  $min_{SegLineLength}$  масштабируется линейно (листинг 3).

Листинг 3 — Метод *findMinAllowedLineSegmentLength*

```
1. double findMinAllowedLineSegmentLength(const cv::Mat &image)  
2. {  
3.     double wScale = image.cols / POCKET_IMAGE_WIDTH;  
4.     double hScale = image.rows / POCKET_IMAGE_HEIGHT;  
5.     return std::max(wScale, hScale) * MIN_ALLOWED_LINE_SEGMENT_LENGTH;  
6. }
```

Далее производится подготовка структур данных к последующим этапам работы метода. На основе каждого отрезка сегмента линии, представленного четверкой целых чисел, создается объект класса *LineSegment* и вычисляются его поля. Описание полей типа *LineSegment* приведено в таблице 7.

Таблица 7 — Описание полей типа *LineSegment*.

| Поле          | Тип              | Описание  |
|---------------|------------------|---|
| <i>origin</i> | <i>cv::Vec4i</i> | Оригинальная четверка координат концов в нотации $(x_1, y_1, x_2, y_2)$ . |

| Поле   | Тип         | Описание   |
|--------|-------------|--|
| from   | cv::Point3d | Координаты точки начала отрезка в нормализованных координатах.                                       |
| to     | cv::Point3d | Координаты точки конца отрезка в нормализованных координатах.  |
| middle | cv::Point3d | Середина отрезка в нормализованных координатах.  |
| line   | cv::Point3d | Координаты линии, которую задает сегмент, в нормализованных координатах, вычисленная по формуле (4). |

На основе полученных сегментов линий, производится поиск точек схождения перспективы.

### 3.2.2 Вычисление точек схождения перспективы

Ранее этапы уточнения и определения направления ТСП были описаны следующими после этапа кластеризации. Однако все три этапа независимы для разных ТСП, и потому достаточно проводить их последовательно только в рамках одной ТСП, что и осуществлено в текущей реализации приложения.

Для кластеризации используются методы пространства имен `ransac`, являющегося подпространством имен `sgycam`, тестового приложения. Список методов данного пространства имен приведен в таблице 8.

Таблица 8 — Описание методов пространства имен `ransac`.

| Метод | Описание |
|-------|----------|
|       |          |

| Метод  | Описание   |
|--|--|
| <pre>int countInducedSegments(     const std::vector&lt;LineSegment&gt; &amp;segments,     const std::set&lt;int&gt; &amp;notUsed,     const cv::Point3d &amp;vp,     double angleEpsilon )</pre>                  | Подсчет индуцированных сегментов линий относительно точки vp с допустимой ошибкой угла angleEpsilon. Поиск ведется на множестве сегментов segments с учетом множества-индикатора notUsed (т.е. только среди тех, чьи индексы имеются в этом множестве).                    |
| <pre>std::vector&lt;int&gt; getInducedSegments(     const std::vector&lt;LineSegment&gt; &amp;segments,     const std::set&lt;int&gt; &amp;notUsed,     const cv::Point3d &amp;vp,     double angleEpsilon )</pre> | Возвращает список индексов индуцированных сегментов линий относительно точки vp с допустимой ошибкой угла angleEpsilon. Поиск ведется на множестве сегментов segments с учетом множества-индикатора notUsed (т.е. только среди тех, чьи индексы имеются в этом множестве). |
| <pre>std::vector&lt;LineSegment&gt; resolveIndices(     const std::vector&lt;LineSegment&gt; &amp;segments,     const std::vector&lt;int&gt; &amp;indices )</pre>  | Вспомогательный метод, разыменовывающий список индексов сегментов в список самих объектов сегментов линий типа LineSegment.  |
| <pre>void markInducedSegmentsAsUsed(     std::set&lt;int&gt; &amp;notUsed,     const std::vector&lt;int&gt; &amp;toErase )</pre>   | Вспомогательный метод, удаляющий из множества-индикатора notUsed список индексов СЛ toErase.   |

| Метод   | Описание   |
|---|--|
| <pre>std::vector&lt;LineSegment&gt; nextCluster(     const std::vector&lt;LineSegment&gt; &amp;segments,     const std::set&lt;int&gt; &amp;notUsed,     cv::Point3d &amp;outVanishingPoint )</pre> | Поиск наилучшего кластера на множестве сегментов segments с учетом множества–индикатора notUsed. Метод также возвращает координаты ТСП через аргумент outVanishingPoint. |

В качестве допустимой погрешности угла  $\epsilon_\alpha$  функции–индикатора принадлежности сегмента кластеру (24) опытным путем выбрано значение  $\epsilon_\alpha = 0,04$  рад  $\cong 2,3$  градуса.

После выделения первого кластера посредством вызова метода `nextCluster`, для него производится уточнение направления ТСП в методе `ImageProcessor::refineVanishingPoint`, который приведен в листинге 4.

*Листинг 4 — Метод `ImageProcessor::refineVanishingPoint`.*

```

1. cv::Point3d ImageProcessor::refineVanishingPoint(const
   std::vector<LineSegment> &cluster)

2. {
3.     cv::Mat A = cv::Mat::zeros(cluster.size(), 3, CV_64FC1);
4.     for (int i = 0; i < cluster.size(); i++)
5.         setRow(A, i, toNormalized(cluster[i]));
6.
7.     cv::Mat res = cv::Mat::zeros(3, 1, CV_64FC1);
8.     solveZ(A, res);
9.     res = res.t();
10.    return cv::Point3d(res);
11. }
```

В данной процедуре для решения задачи минимизации (32) используется существующая реализация метода сингулярного разложения матрицы

библиотеки OpenCV, которой отвечает класс `cv::SVD`. Доступ к матрицам разложения производится через обращение к соответствующим полям объекта данного класса — `u`, `w` и `vt` — которые являются объектами типа данных `Mat` библиотеки OpenCV.

Матрица `w` в данной реализации является матрицей-столбцом, которой соответствует значения диагонали матрицы  $\Sigma$  сингулярного разложения, то есть — сингулярные числа. В таком случае решением задачи минимизации матрицы  $A_L^T$  уравнения (32) будет вектор-строка матрицы `vt`, соответствующий последнему значению матрицы `w`. В листинге 5 приведен метод `ImageProcessor::SolveZ`, в котором производится вычисление вектора направления ТСП путем решения описанной задачи.

*Листинг 5 — Метод `ImageProcessor::SolveZ`.*

```

1. void ImageProcessor::solveZ(const cv::Mat &a, cv::Mat &res)
2. {
3.     cv::SVD svd(a, cv::SVD::FULL_UV);
4.     if (settings.TRACE_ENABLED)
5.     {
6.         std::cout << "SVD results:" << std::endl;
7.         std::cout << "W: " << svd.w << std::endl;
8.         std::cout << "Vt: " << svd.vt << std::endl;
9.     }
10.    res = svd.vt.row(svd.w.rows - 1);
11. }
```

После уточнения вектор направления  $d_{vp}$  переводится обратно в ненормированные координаты изображения:

$$vp = K * d_{vp} \quad (39)$$

Полученная ТСП является уточнением точки, на основе которой был

индуцирован кластер соответствующих ей сегментов, и, соответственно, может быть использована для уточнения самого кластера. Затем процедура уточнений ТСП и кластера повторяется.

Пример результата работы этапов кластеризации и уточнения приведены на рис. 26. На изображении помечены цветами три найденных кластера СЛ. Темными вариантами цветов помечены сегменты, входившие в первоначальные кластеры СЛ, но отброшенные на этапах уточнения. Перечеркнутыми кругами помечены найденные ТСП, попавшие в рамки кадра: темным цветом — грубая оценка, ярким — уточненная.

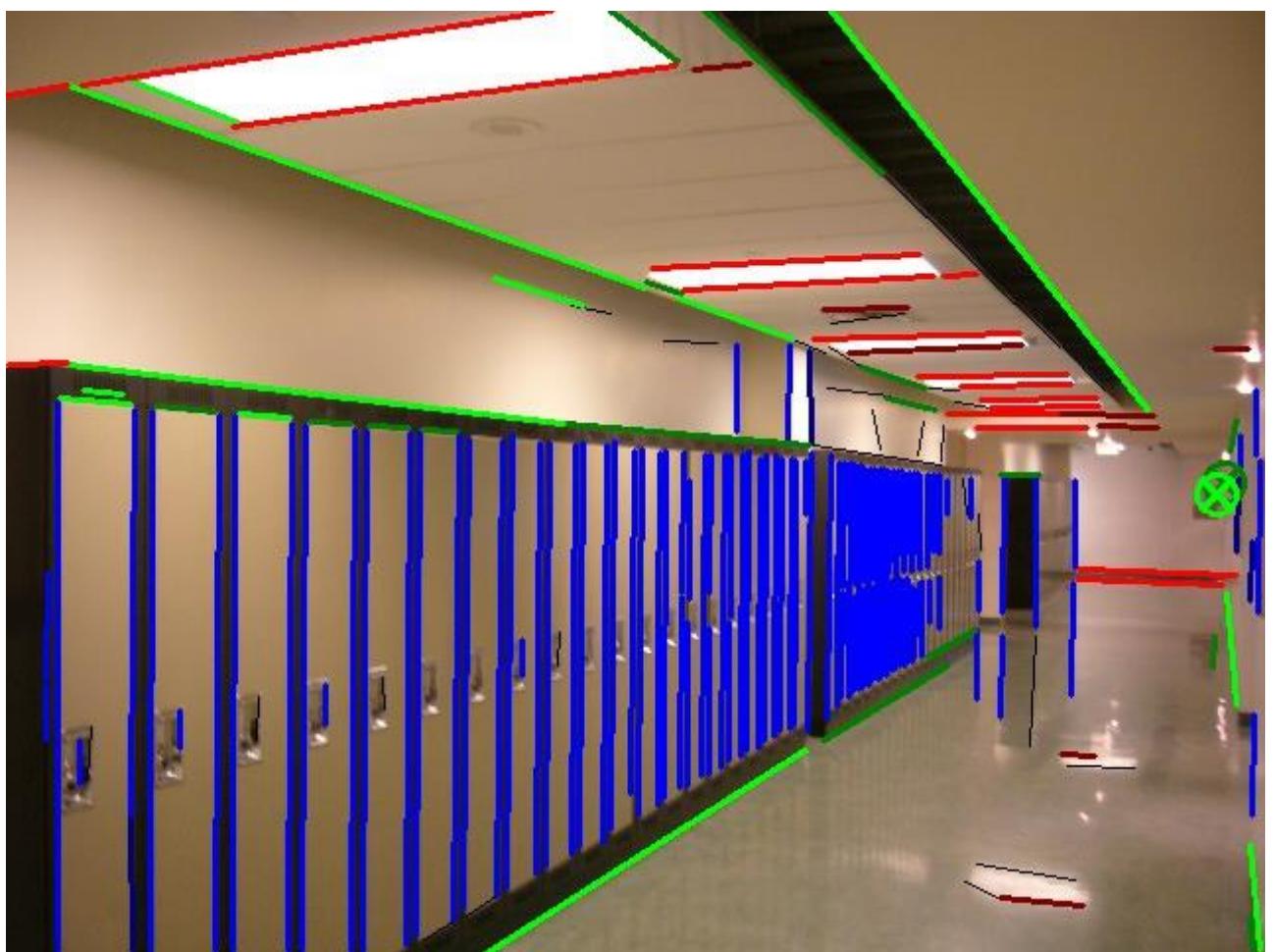


Рисунок 26 — Пример результата работы этапов кластеризации СЛ и уточнения направлений ТСП.

Всего производится две итерации уточнения, которые описаны в листинге 5 метода `ImageProcessor::smartVpRefinement`. Заметим, что эта

процедура производится с меньшим углом толерантности  $\epsilon_\alpha(i) = \frac{\epsilon_\alpha}{4-i}$ , зависящим от номера итерации.

Подобная эвристика позволяет повысить точность определения координат направления ТСП. При этом достигается некоторая гибкость, когда мы можем специально ослабить ограничения на допустимый угол при первоначальном выборе кластера, усилив ограничения на допустимый угол при последующих уточнениях.

Первое сократит вероятность случая, когда из-за неудачного случайного выбора внутренних сегментов оценка ТСП будет чересчур грубой и приведет к выделению лишь некоторой части кластера. Так как обычно на изображении кластеры искомых ТСП достаточно хорошо отличимы друг от друга, ослабление первоначального допустимого угла в худшем случае приведет к захвату шумовых сегментов, которые будут отброшены на этапах уточнения.

### *Листинг 5 — Метод ImageProcessor::smartVpRefinement*

```

1. cv::Point3d ImageProcessor::smartVpRefinement(const
   std::vector<LineSegment> &originCluster, int i)

2. {
3.     int steps = 2;
4.     cv::Point3d vp, normalizedVp;
5.     std::vector<int> indices;
6.     std::vector<LineSegment> cluster = originCluster;
7.     do
8.     {
9.         normalizedVp = refineVanishingPoint(cluster);
10.        cv::Point3d t(normalizedVp);
11.        vp = fromNormalized(cv::Mat(t));
12.        normalizeZ(vp);
13.        indices = ransac::getInducedSegments(
14.            segments, notUsedSegments, vp,
15.            ANGLE_EPSILON / (4 - steps)
16.        );

```

```

17.         cluster = ransac::resolveIndices(segments, indices);
18.         --steps;
19.     }
20.     while (steps > 0 && cluster.size() > 3);
21.
22.     ransac::markInducedSegmentsAsUsed(notUsedSegments, indices);
23.     drawInducedCluster(vp, cluster, colors[i], settings.BUILD_IMAGE);
24.     return normalizedVp;
25. }
```

Второе позволяет прямо повысить точность определения ТСП (до некоторого предела, конечно).

В результате описанных этапов для выделенного первого кластера СЛ вычисляется уточненное направление ТСП  $d_{vp}$ . Затем процесс кластеризации и уточнения повторяется еще два раза для оставшихся направлений ТСП.

### 3.2.3 Уточнение матрицы поворота на основе полученных направлений ТСП

Полученные направления ТСП становятся векторами–стоблцами матрицы  $D$ , которую для начала требуется проанализировать на предмет заведомо ложных направлений, то есть направлений, не являющимися взаимно ортогональными к остальным двум. Поиск и обнуление ложных направлений производится в методе `ImageProcessor::refineVpBasis`, исходный код которого приведен на листинге 6.

*Листинг 6 — Метод `ImageProcessor::refineVpBasis`.*

```

1. void ImageProcessor::refineVpBasis(cv::Mat &vpBasis)
2. {
3.     cv::Mat c = vpBasis;
4.     cv::Mat r = vpBasis.t();
```

```

5.     for (int i = 0; i < 3; i++)
6.     {
7.         auto yi = (i+1)%3;
8.         auto zi = (i+2)%3;
9.         double a1 = abs(angleBetween(r.row(i), c.col(yi)) - CV_PI/2);
10.        double a2 = abs(angleBetween(r.row(i), c.col(zi)) - CV_PI/2);
11.        if (min(a1, a2) > 0.2)
12.        {
13.            vpBasis.col(i) = 0 * vpBasis.col(i);
14.            break;
15.        }
16.        if (max(a1, a2) > 1.0)
17.        {
18.            int col = a1 >= a2 ? yi : zi;
19.            vpBasis.col(col) = 0 * vpBasis.col(col);
20.            break;
21.        }
22.    }
23. }

```

Затем над этой матрицей проводится процесс ортогонализации (33). Для этой цели снова используется реализация метода сингулярного разложения матрицы `cv::SVD` библиотеки OpenCV. В листинге 7 представлен метод `getNearestOrthogonalMatrix`, на котором показано использование класса `cv::SVD`.

#### *Листинг 7 — Метод `getNearestOrthogonalMatrix`.*

```

1. cv::Mat getNearestOrthogonalMatrix(const cv::Mat &a)
2. {
3.     cv::SVD svd(a, cv::SVD::FULL_UV);
4.     return svd.u * svd.vt;
5. }

```

Чтобы получить матрицу поворота из матрицы  $\widehat{D}$ , столбцы полученной матрицы необходимо переупорядочить и проверить их ориентацию в

соответствии с процедурой, описанной в 2.6.1 . Эта операция производится в три последовательных вызова функции `reorderColumn` — по запуску для каждой из координат векторов. Тело метода приведено в листинге 8.

Полученная матрица после транспонирования становится искомой матрицей поворота  $R$  (34).

*Листинг 8 — Метод `reorderColumn`.*

```

1. void reorderColumn(cv::Mat &a, int row)
2. {
3.     double m;
4.     int i;
5.     findMaxOnRow(a.row(row), m, i);
6.     swapColumns(a, row, i);
7.     if (a.at<double>(row, row) < 0)
8.         a.col(row) = -1 * a.col(row);
9. }
```

Ниже приведены две таблицы, в которых приведен пример результатов работы на данном этапе. Таблица 9 содержит матрицу  $D$ , полученную из векторов-столбцов направлений ТСП из предыдущих этапов вычислений. Таблица 10 содержит матрицу  $\widehat{D}$ , полученную в результате выполнения текущего этапа. Соответствие направлений ТСП из таблицы 9 и осей координат показано добавленными верхними индексами.

Таблица 9 — Пример матрицы  $D$ .

| $vp_1$       | $vp_2$       | $vp_3$       |
|--------------|--------------|--------------|
| 0,006024762  | 0,919248728  | 0,394873420  |
| -0,999897274 | 0,026661950  | -0,031347850 |
| -0,013005493 | -0,392773366 | 0,918200574  |

Таблица 10 — Пример матрицы  $\widehat{D}$ .

| $vp_2^x$     | $vp_1^y$     | $vp_3^z$     |
|--------------|--------------|--------------|
| 0,919037357  | -0,009094845 | 0,394065501  |
| 0,018670735  | 0,999616054  | -0,020473119 |
| -0,393728002 | 0,026173054  | 0,918854304  |

### 3.2.4 Вычисление углов Эйлера по известной матрице поворота

Вычисление углов Эйлера в нотации (1,2,3) производится в соответствии с формулой (37) и приведено в листинге 9 метода `getEulerAngles`.

*Листинг 9 — Метод `getEulerAngles`.*

```

1. cv::Mat getEulerAngles(const cv::Mat &r)
2. {
3.     cv::Mat a = cv::Mat::eye(1, 3, CV_64FC1);
4.     a.at<double>(0, 0) = atan2(r.at<double>(1,2), r.at<double>(2, 2));
5.     a.at<double>(0, 1) = -asin(r.at<double>(0,2));
6.     a.at<double>(0, 2) = atan2(r.at<double>(0,1), r.at<double>(0, 0));
7.
8.     a = a * 180 / CV_PI;
9.     return a;
10. }
```

Полученные результаты сохраняются на диск в соответствии с глобальной и специфичной конфигурацией запуска и возвращаются в виде объекта структуры `SingleRunResult`, которая содержит поля, приведенные в таблице 11.

Таблица 11 — Описание полей структуры `SingleRunResult`.

| Поле                 | Тип                  | Описание  |
|----------------------|----------------------|---|
| <code>vpBasis</code> | <code>cv::Mat</code> | Матрица единичных векторов—столбцов направлений найденных |

| <b>Поле</b>  | <b>Тип</b> | <b>Описание</b>  |
|--------------|------------|--|
|              |            | ТСП до анализа и процесса ортогонализации  |
| orthoVpBasis | cv::Mat    | Матрица единичных векторов-столцов направлений найденных ТСП после процесса ортогонализации. |
| eulerAngles  | cv::Mat    | Вектор-столбец углов Эйлера в нотации (1, 2, 3) в градусах.                                  |
| runtime      | double     | Время работы процесса обработки в миллисекундах.   |

На рис. 27 приведен пример результатов работы этапа вычисления углов Эйлера на изображении, снятом внутри помещения.

На рис. 28 приведено пример результата работы приложения в целом. На рисунке слева — оригинальное изображение, справа — изображение, полученное в результате вычисления углов ориентации. Результирующие углы ориентации камеры равны: крен  $\cong 2,0$  градуса , тангаж  $\cong -26,8$  градуса , рыскание  $\cong -0,9$  градуса.



Рисунок 27 — Результат работы этапа вычисления углов Эйлера в нотации (1, 2, 3) на примере кадра, снятого внутри помещения. Сверху на изображении выведена строка со значениями полученных углов: крен  $\cong 1,6$  градуса, тангаж  $\cong 23,2$  градуса, рыскание  $\cong 1,2$  градуса.



Рисунок 28 — Пример результата работы приложения на снимке вне помещения.

### **3.3 Блок управления процессом тестирования наборов изображений**

Второй режим работы приложения связан с тестированием наборов изображений. Управление этим процессом производится с использованием класса `YorkUrbanDbTester`. Данный класс содержит единственный метод `run`, созданный для этих целей.

Алгоритм тестирования состоит из следующих шагов:

1. Чтение списка изображений
2. Инициализация векторов средней и среднеквадратичной ошибки определения направлений ТСП, переменной времени работы тестирования.
3. Для каждого изображения из списка выполняются шаги 4–7
4. Создается и инициализируется объект `ImageProcessor` со специфичной конфигурацией текущего изображения
5. Производится запуск процесса обработки изображения, результатом которого являются матрицы направлений ТСП, углы ориентации и время работы.
6. Производится чтение файлов матриц «верных» значений направлений ТСП.
7. Определяются углы между соответствующими полученными направлениями ТСП и верными, которые задают вектор отклонения (ошибки). Результат сохраняется в субдиректории изображения.
8. По полученными векторам ошибок каждого изображения вычисляются векторы средней и среднеквадратичной ошибки.
9. Результаты ошибок каждого изображения, а также векторы средней и среднеквадратичной ошибки сохраняются в виде отчета в корневой директории тестового набора изображений.

10. В поток стандартного вывода печатается статистика по времени выполнения тестирования — общее время в секундах и среднее количество обработанных кадров в секунду (fps).

## ГЛАВА 4. ТЕСТИРОВАНИЕ ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

### 4.1 *Методология разработки и используемые средства*

В рамках данной дипломной работы создано тестовое Win32 консольное приложение, в котором реализован алгоритм определения углов пространственной ориентации камеры с помощью нахождения точек схождения перспективы (ТСП) на изображении, а также векторов единичных направлений, соответствующих им.

В качестве языка программирования был выбран объектно-ориентированный язык высокого уровня C++, который является одним из стандартов де-факто в работах по компьютерному зрению наряду с языком Python и средой MatLab. Разработка велась с использованием средств интегрированной среды разработки Visual Studio 2010.

#### 4.1.1 Библиотека алгоритмов компьютерного зрения OpenCV

При разработке приложения активно использовалась библиотека алгоритмов компьютерного зрения с открытым исходным кодом OpenCV версии 3.0.0 beta [23] по следующим причинам:

- Содержит широкий набор структур данных для работы с геометрическими и алгебраическими примитивами, такими как точки, матрицы, с поддержкой математических операций над ними

- Содержит готовые реализации части алгоритмов, которые написаны компетентными в данной области разработчиками с использованием низкоуровневых и платформозависимых оптимизаций, а также протестированы и проверены временем. Были использованы детектор сегментов линий Джии, которому соответствует класс `cv::LineSegmentDetector`, и реализация метода сингулярного разложения матриц, которой соответствует класс `cv::SVD`.

При подключении объектов данной библиотеки к проекту возникли проблемы, поэтому от поставляемой скомпилированной библиотеки пришлось отказаться. Вместо этого были скачаны файлы исходного кода библиотеки, которые были собраны вручную с использованием утилиты CMake [24] и компилятора C++, поставляемого со средой разработки Visual Studio 2010.

## **4.2 Руководство пользователя**

### **4.2.1 Требования для сборки и работы с приложением**

Для использования данного приложения требуется установленная на компьютере пользователя библиотека компьютерного зрения OpenCV версии 3.0.0 beta [23]. Описание процесса установки и необходимые файлы можно найти на сайте библиотеки [25]. Для успешного подключения необходимо наличие переменной окружения `$(OPENCV_DIR)`.

Исходный код приложения объединен в проект VisualStudio 2010, поэтому для компиляции кода данная среды разработки также потребуется установленной. Другой вариант — воспользоваться утилитой msbuild.

### **4.2.2 Режимы работы и входные аргументы**

Приложение запускается с набором обязательных и опциональных аргументов. Часть аргументов является порядко-зависимыми, то есть

интерпретируются в зависимости от того, в каком порядке следуют. Остальные аргументы являются предварительно заданными строковыми ключами запуска, которые начинаются со знака минус «`-`» и не зависят от позиции во входном массиве аргументов. Входные аргументы первой категории будем называть неименованными, а второй — именованными.

В зависимости от набора входных аргументов приложение может работать либо в режиме обработки одного изображения, либо в режиме тестирования директории с набором изображений.

Ниже приведен список неименованных аргументов в порядке их следования:

1. Единственный обязательный аргумент — строка абсолютного или относительного пути к входному файлу изображения (для режима обработки одного изображения) или к корневой директории набора изображений для тестирования (подробнее в 4.2.4 ).
2. Необязательный аргумент — строка абсолютного или относительного пути к текстовому файлу с матрицей калибровки камеры. Формат файла описан далее в данной главе. В случае, если путь не задан, в качестве матрицы калибровки будет использована матрица с параметрами  $\frac{f}{m_x} = \frac{f}{m_y} = 1$  и главной точкой в центре изображения.

Далее приведены именованные строковые ключи запуска и то, как они влияют на работу приложения:

- «`-noWait`» — выключает интерактивный режим ожидания ввода с клавиатуры пользователя после завершения работы, закрывая все окна и завершая работу приложения. Такой режим полезен при массовом запуске приложения, например, с помощью `.bat` скриптов.
- «`-trace`» — включает вывод трассировки работы приложения в стандартный поток вывода.

– «`-noShow`» — выключает вывод окна с обработанным изображением, на котором отрисованы разными цветами кластеры сегментов линий, точки схождения перспективы (в случае, если их координаты попадают в область изображения) и результат вычисления углов ориентации. Стоит отметить, что само изображение все-равно будет построено и сохранено в файл. Для полной отмены построения изображения-результата используется ключ «`-noImage`».

– «`-noRaw`» — выключает отрисовку на изображении-результате всего множества сегментов линий, прошедших этап фильтрации.

– «`-noImage`» — полностью выключает построение изображения-результата. Как следствие, не будет отображено окно с результирующим изображением (как при действии ключа «`-noShow`») и не будет сохранен файл с ним. Данный ключ имеет смысл для замеров скорости работы алгоритма, так как в противном случае построение изображения несколько искажает результаты скорости в сторону увеличения. Стоит отметить, что время сохранения результирующего изображения в файл в замере скорости не участвует — только построение изображения в оперативной памяти приложения.

– «`-size640`» — включает автоматическое масштабирование обрабатываемых изображений в сторону уменьшения до размеров  $640 \times 480$ . В случае, если стороны исходного изображения имеют другую пропорцию, коэффициент масштабирования выбирается так, чтобы одна из сторон была равна соответствующей стороне прямоугольника  $640 \times 480$ , а другая не превышала. Стоит отметить, что именно этот размер отвечает истинному значению булева поля `POCKET_SIZE` структуры `Settings`.

– «`-size1280`» — работает аналогично ключу «`-size640`», но соответствует ложному значению булева поля `POCKET_SIZE` структуры

Settings.

- «`-yorkUrbanDb`» — включает режим тестирования базы данных изображений. Иначе приложение работает в режиме обработки одного изображения.

Стоит также отметить, что именованные строковые ключи не зависят от регистра, поэтому их можно вводить в любом удобном виде.

В листинге 10 приведены примеры строки аргументов запуска приложения.

*Листинг 10 — Примеры строк аргументов запуска приложения.*

```
1. >Gyrocam.exe -size640 -trace "TestSamples/urban.jpg"  
2. >Gyrocam.exe "TestSamples/YorkUrbanDB_indoor/" -noImage -yorkUrbanDb
```

#### 4.2.3 Обычный режим работы

Для изображения, заданного входным аргументом, приложение рассчитывает матрицу единичных векторов направлений ТСП, углы ориентации камеры и формирует изображение—результат (если не указан ключ `-noImage`). Поддерживаются следующие форматы изображений: tiff, jpg, png.

Изображение формируется в процессе работы метода следующим образом:

- В качестве основы берется исходное изображение (возможно, отмасштабированное в соответствии с ключами запуска).
- Далее черным цветом отрисовываются выделенные детектором сегментов линий на изображении все сегменты, прошедшие этап фильтрации по длине.
- Затем по мере выделения кластеров сегментов линий они отрисовываются темными вариантами базовых цветов (в половину

яркости чистого цвета). Базовые цвета идут в следующем порядке: синий, зеленый, голубой. Отрисовка производится на этом этапе, чтобы можно было оценить характер и качество первоначальной кластеризации.

- Теми же цветами в виде перечеркнутых кругов отрисовываются точки схождения перспективы в случае, если их координаты попадают в область изображения.
- Кластеры СЛ, полученные после этапа вычисления направлений ТСП, отрисовываются на изображении яркими вариантами своих базовых цветов (в полную яркость). Стоит отметить, что цвета задают номер ТСП, поэтому из-за перестановки векторов направлений неуточненный и уточненный кластеры одного базового цвета могут соответствовать разным направлениям.
- Таким же образом отрисовываются уточненные оценки самих ТСП.

Пример получающегося выходного изображения приведен на рис. 29, на котором ошибка определения векторов горизонтальных направлений ТСП на данном тесте не превышает одного градуса, а вертикального — половины градуса.

В случае, если не указан ключ «`-noImage`», изображение сохраняется в той же директории, что и оригинал, с именем `«$name$_gyrocam_processed.$ext$»`, где `$name$` — имя исходного изображения, а `$ext$` — его расширение.

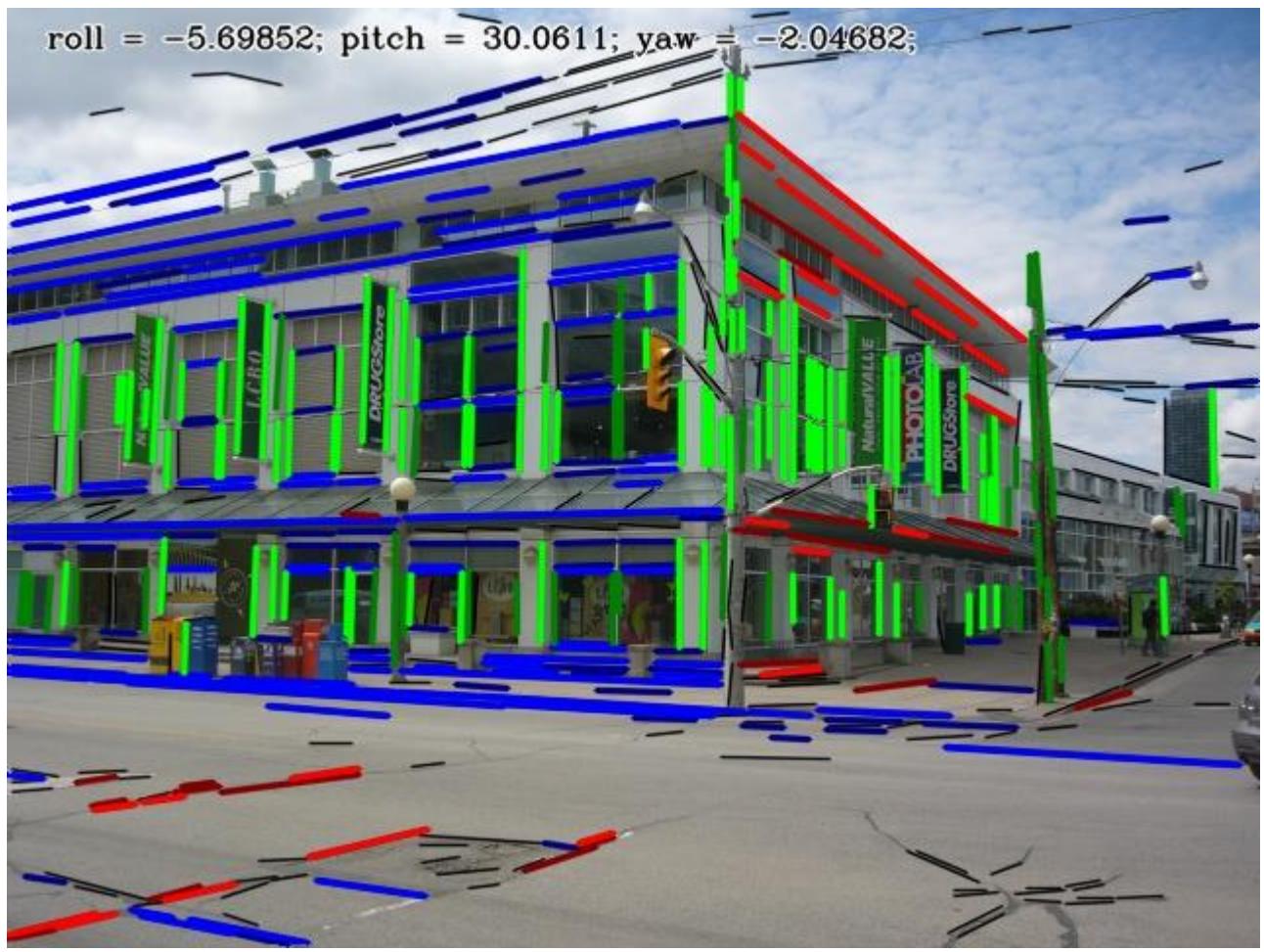


Рисунок 29 — Пример отрисовки изображения—результата в результате работы приложения.



Рисунок 30 — Пример окна вывода с обработанным изображением.

На рис. 30 приведен пример окна вывода полученного изображения.

Выведенное пользователю изображение также сохраняется на диск в выходной файл. Путь, название и расширение файла определяется соответствующим (вторым) входным аргументом командной строки.

Помимо изображения, в выходные файлы сохраняются матрицы направлений ТСП — до анализа и ортогонализации и после — а также вектор углов Эйлера полученной ориентации камеры. По аналогии с выходным файлом изображения, имена этих текстовых файлов генерируются на основе имени оригинального изображения:

- «\$name\$ \_gyrocam\_angles.txt» — для вектора углов,

- «\$name\$\_gyrocam\_vp\_basis.txt» — для неортогонализированной матрицы направлений ТСП,
- «\$name\$\_gyrocam\_vp\_ortho\_basis.txt» — для ортогонализированной матрицы направлений.

Формат вывода матриц направлений ТСП в файл приведен в листинге 11. Границы матрицы задаются квадратными скобками, строки матрицы разделяются точкой с запятой с последующим переводом строки, числа в рамках одной строки разделяются запятыми.

*Листинг 11 — Пример форматированного вывода матрицы направлений ТСП в текстовый файл.*

```

1. [0.99999738, 0.0022501017, 0.00047868371;
2. 0.93076748, 0.36539984, -0.012442469;
3. -0.10985497, -0.99394757, -0.00042265668]
```

На рис. 31 приведен пример окна стандартного вывода после работы приложения с включенным ключом трассировки.

```
C:\Windows\System32\cmd.exe - Gyrocam.exe ..../TestSamples/urban1.jpg" -trace
p:\Projects\Study\Graduate work\Gyrocam\Gyrocam\Release>Gyrocam.exe "....\TestSamples\urban1.jpg" -trace
Raw segments: 752; passed length filter: 319
SVD results:
w: [84426.57167872004;
 371.284914349213;
 5.838691622512362]
vt: [-0.0009192072208513084, 0.005624161063385908, 0.9999837618033696;
 0.05772869239591016, -0.9983162192285152, 0.005667847875660545;
 0.9983318852627837, 0.05773296491274582, 0.0005929840133945687]
SolveZ Manually:
[0.9983318852627837;
 0.05773296491274582;
 0.0005929840133945687]
SolveZ:
[0.9983318852627837;
 0.05773296491274582;
 0.0005929840133945687]
SolveZ diff: 0
SVD results:
w: [83241.91680116508;
 306.5949218934897;
 3.133971457386271]
vt: [-0.0009370580530902272, 0.005827500857307915, 0.9999825809262696;
 0.05475428757956974, -0.9984826038986909, 0.005870068427499057;
 0.9984994190853888, 0.05475883440548951, 0.0006165555070425412]
SolveZ Manually:
[0.9984994190853888;
 0.05475883440548951;
 0.0006165555070425412]
SolveZ:
[0.9984994190853888;
 0.05475883440548951;
 0.0006165555070425412]
SolveZ diff: 0
OriginVP1
[2278.57, 356.429, 1]
RefinedVP1
[0.998499, 0.0547588, 0.000616556]
SVD results:
w: [34086.54283719259;
 216.294535462039;
 4.804390109111899]
vt: [0.002166823140821669, 9.985185459050287e-005, 0.9999976474507746;
 -0.9998336787408241, -0.01810838023517839, 0.002168276008394634;
 0.0181085541407032, -0.9998360248535082, 6.059758976793406e-005]
SolveZ Manually:
[0.0181085541407032;
 -0.9998360248535082;
 6.059758976793406e-005]
SolveZ:
[0.0181085541407032;
 -0.9998360248535082;
 6.059758976793406e-005]
SolveZ diff: 0
SVD results:
w: [26773.78519409887;
 175.7647106339144;
 1.773907452515403]
vt: [0.001315567367647895, 9.174280981662469e-005, 0.9999991304325008;
 -0.9999373020335218, -0.01112019843683865, 0.00131650622729052;
 0.01112030954705594, -0.9999381644731749, 7.710768754373612e-005]
SolveZ Manually:
[0.01112030954705594;
 -0.9999381644731749;
 7.710768754373612e-005]
SolveZ:
[0.01112030954705594;
 -0.9999381644731749;
 7.710768754373612e-005]
SolveZ diff: 0
OriginVP2
[621, -11604, 1]
RefinedVP2
[0.0111203, -0.999938, 7.71077e-005]
SVD results:
w: [55476.01728692622;
 114.0929813955968;
 2.787877258247233]
```

Рисунок 31 — Пример вывода в консоль в результате работы приложения с включенным режимом трассировки.

#### 4.2.4 Режим тестирования набора изображений

В режиме тестирования набора изображений в качестве первого неименованного входного аргумента ожидается путь к корневой директории

набора. В данной директории должен находиться текстовый файл с именем «`Manhattan_Image_DB_Names.txt`» со списком названий субдиректорий, которые требуется обработать.

Каждая субдиректория из списка, должна содержать файл изображения в формате «`.jpg`». Обозначим имя файла изображения без учета расширения файла `$name$`. Тогда в той же субдиректории должны содержаться текстовый файл «`[$name$GroundTruthVP_Orthogonal_CamParams.mat.txt]`» матрицы верных ортогонализированных единичных направлений ТСП. Формат файла приведен в листинге 12.

*Листинг 12 — Пример формата текстового файла матрицы верных ортогонализированных единичных направлений ТСП*

```
1. orthogonal vp
2. 0.786357903164 0.011432518697 -0.617665399425
3. -0.000796843219 0.999846686902 0.017491933318
4. 0.617770680084 -0.013262737521 0.786246454123
```

Такие необычные с виду названия продиктованы тем, что тестирование проводилось на наборе изображений базы данных YorkUrbanDb, в которой и принят такой формат организации и именования файлов.

По завершению работы приложения в режиме тестирования в корневой директории набора будет сохранен файл со списком векторов угловых отклонений соответствующих направлений полученных ТСП и верных для каждого из протестированных изображений. Последние две строки файла содержат вектор углов средней величины отклонения и вектор углов среднеквадратичной величины отклонения. Все углы указываются в радианах. Для корневой директории `$root$` полный путь к файлу будет иметь вид «`$root$/gyrocam_report.txt`».

Также для каждого из протестированных изображений в его субдиректории с именем `$name$` сохраняются следующие файлы:

- «\$name\$\_gyrocam\_angles.txt» — вектор полученных углов Эйлера в нотации (1, 2, 3) в градусах;
- «\$name\$\_gyrocam\_compare\_vp\_basis.txt» — вектор углов отклонения соответствующих полученных направлений ТСП от верных;
- «\$name\$\_vp\_basis.txt» — матрица неортогонализированных единичных направлений ТСП;
- «\$name\$\_vp\_ortho\_basis.txt» — матрица ортогонализированных единичных направлений ТСП;
- «\$name\$\_gyrocam\_processed.jpg» — обработанное изображение с отрисованными неуточненными и уточненными кластерами СЛ и значениями полученных углов ориентации камеры.

Помимо сохранения файлов, по завершению работы в окно стандартного вывода будет напечатано общее затраченное время на обработку изображений в секундах, а также среднее количество обработанных изображений в секунду (fps).

### **4.3 Тестирование приложения**

#### **4.3.1 База данных изображений YorkUrbanDB**

Описанное консольное приложение было протестировано на наборе изображений YorkUrbanDB (The York Urban Line Segment Database) [2]. Набор состоит из 47 изображений внутри помещений и 55 изображений городских сцен Торонто (Канада).

Для каждого изображения из базы приведены:

- информация о внутренних параметрах камеры
- данные по точкам схождения перспективы, что позволяет оценить

уровень точности работы приложения

- список распознанных сегментов с отмеченным соответствием их точкам схождения перспективы

База данных доступна для скачивания в виде архива по адресу [2].

Данные по каждому изображению хранятся в формате файлов среды MatLab. Чтобы иметь возможность пользоваться ими программно, был написан небольшой скрипт на языке MatLab, осуществляющий конвертацию необходимых данных в текстовый формат. В листинге 13 приведено содержание данного скрипта в сокращении.

*Листинг 13 — Скрипт, осуществляющий конвертацию данных базы данных YorkUrbanDB из формата файлов среды MatLab в текстовое представление.*

```
1. load('Manhattan_Image_DB_Names.mat');
2. numberOfFolders = length(Manhattan_Image_DB_Names);
3. imageNamesFile = fopen('Manhattan_Image_DB_Names.txt', 'w');
4.
5. % Process all image files in those folders.
6. for k = 1 : numberOfFolders
7.     thisFolder = Manhattan_Image_DB_Names{k};
8.     fprintf(imageNamesFile, '%s\r\n', Manhattan_Image_DB_Names{k});
9.
10.    filePattern = sprintf('%s*GroundTruthVP_Orthogonal_CamParams.mat',
11.                           thisFolder);
12.    fileStruct = dir(filePattern);
13.    path = sprintf('%s%s', thisFolder, fileStruct.name);
14.    load(path, 'vp_orthogonal');
15.    outputFile = fopen(sprintf('%s.txt', path), 'w');
16.
17.    fprintf(outputFile, 'orthogonal vp\r\n');
18.    nRows = length(vp_orthogonal);
19.    for row = 1 : nRows
20.        fprintf(outputFile, '%4.12f ', vp_orthogonal(row,:));
21.        fprintf(outputFile, '\r\n');
22.    end
23.    fclose(outputFile);
24.end
```

```
24.  
25. fclose(imageNamesFile);
```

### 4.3.2 Конфигурация тестирования

Тестирование проводилось на личном персональном компьютере Intel Core i7 920, имеющем 4 физических ядра — 8 виртуальных в режиме Hyper-threading. Тактовая частота составляет 2,66—2,80 ГГц. Оперативная память имеет размер 6 Гб и работает в трехканальном режиме.

## 4.4 Результаты тестирования

### 4.4.1 Результаты тестирования метода на изображениях «внутри помещения»

В результате тестирования было обработано 45 снимков внутри помещений из коллекции базы данных YorkUrbanDB [2]. Время и качество обработки всех снимков варьируется от запуска к запуску, поэтому для каждого из изображений было проведено по 20 запусков алгоритма.

Общее время обработки набора изображений в среднем составило:

- $\approx 4,6$  сек или  $\approx 9,8$  fps — для запусков без ключа «`-noImage`», т.е. с формированием изображения–результата.
- $\approx 4,28$  сек или  $\approx 10,5$  fps — для запусков с ключом «`-noImage`».

В качестве меры отклонения от истинных значений было принято решение использовать вектор, составленный из абсолютных углов между соответствующими истинными и полученными направлениями ТСП. Углы измерялись в градусах. По результатам нескольких запусков алгоритма на каждом из изображений вычислялся вектор среднего отклонения, который в дальнейшем использовался как вектор ошибки работы метода на изображении.

Векторы минимальной, максимальной и средней ошибки на наборе

протестированных изображений составили:

- $M_{MAX} = [14,74; 1,92; 14,9]$
- $M_{MIN} = [0,10; 0,08; 0,12]$
- $M_{AVG} = [1,43; 0,58; 1,54]$

Также для каждого изображения по результатам итераций запусков считался вектор среднеквадратичного отклонения. В результате средний вектор среднеквадратичного отклонения на всем наборе составил:

$$V_{AVG} = [0,24; 0,04; 0,25]$$

В итоге был построен график распределения ошибок. Имеются следующие оценки по горизонтальным осям:

- 50 перцентиль  $\cong 1$  градус;
- 75 перцентиль  $< 1,5$  градуса;
- 90 перцентиль  $< 3$  градусов.

И по вертикальной оси:

- 50 перцентиль  $\cong 0,44$  градуса
- 75 перцентиль  $\cong 0,7$  градуса
- 90 перцентиль  $\cong 1$  градус

Результаты распределения полученных направлений представлены на диаграммах в разделе дополнительных иллюстраций (Диаграмма 1, Диаграмма 2). На рис. 32–33 приведены примеры результатов обработки изображений, на которых метод показал наилучшую и наихудшую точность соответственно.

В целом, метод показал себя очень хорошо на данном наборе. Среди изображений есть несколько, на которых величина ошибки значительна (больше 5 градусов). На подавляющем же большинстве изображений текущая реализация метода показала высокий уровень точности, исходя из значения вектора средней ошибки  $M_{AVG}$  и графика плотности распределения ошибки.

Также, исходя из значения вектора среднеквадратичного отклонения  $V_{AVG}$ , можно отметить высокий показатель стабильности работы при последовательных запусках на одном и том же изображении.

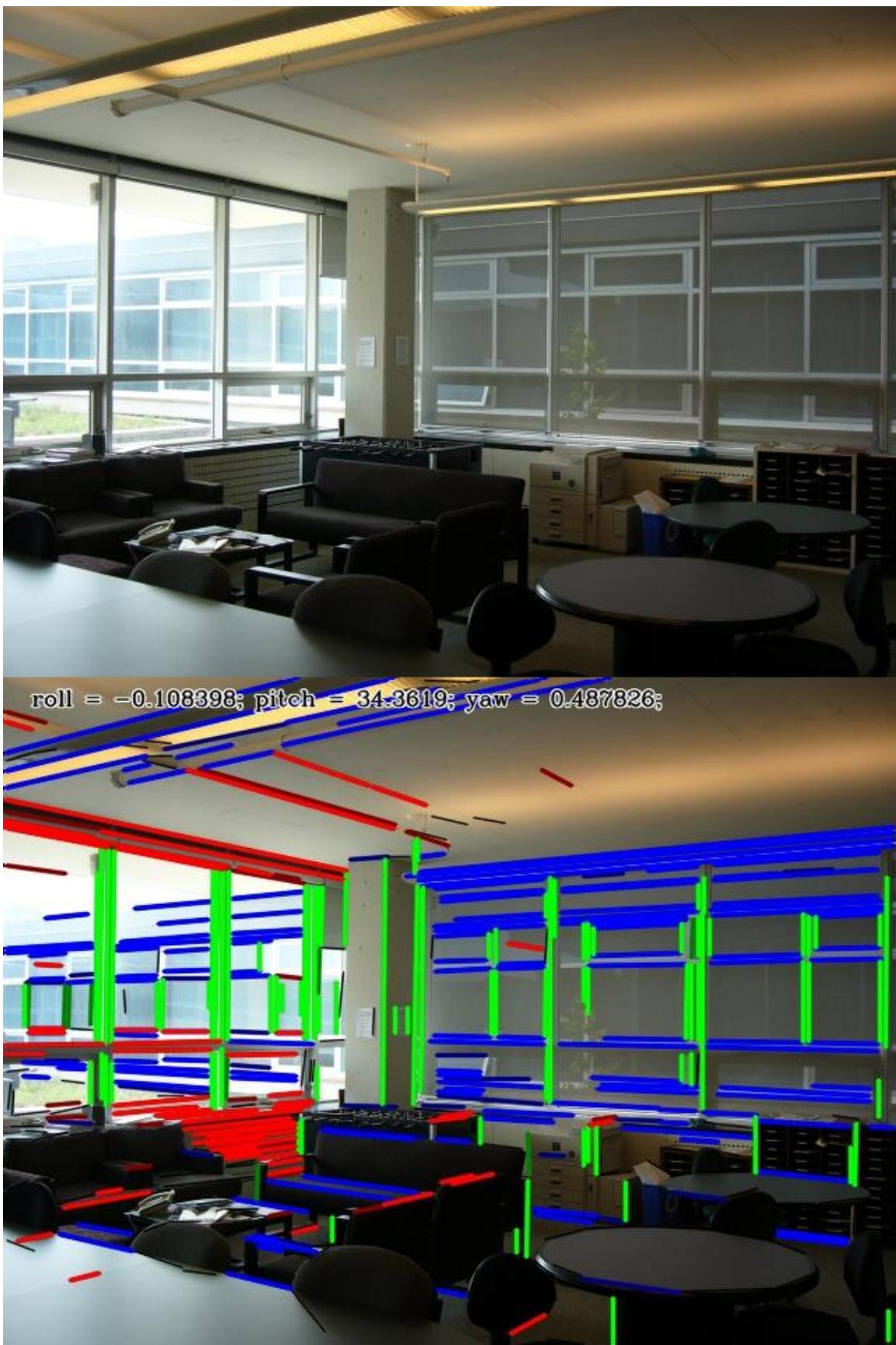


Рисунок 32 — Пример обработки изображения, на котором метод показал наилучшую точность среди набора изображений «внутри помещения». Средний вектор ошибки направлений ТСП в градусах — (0,10; 0,08; 0,12).

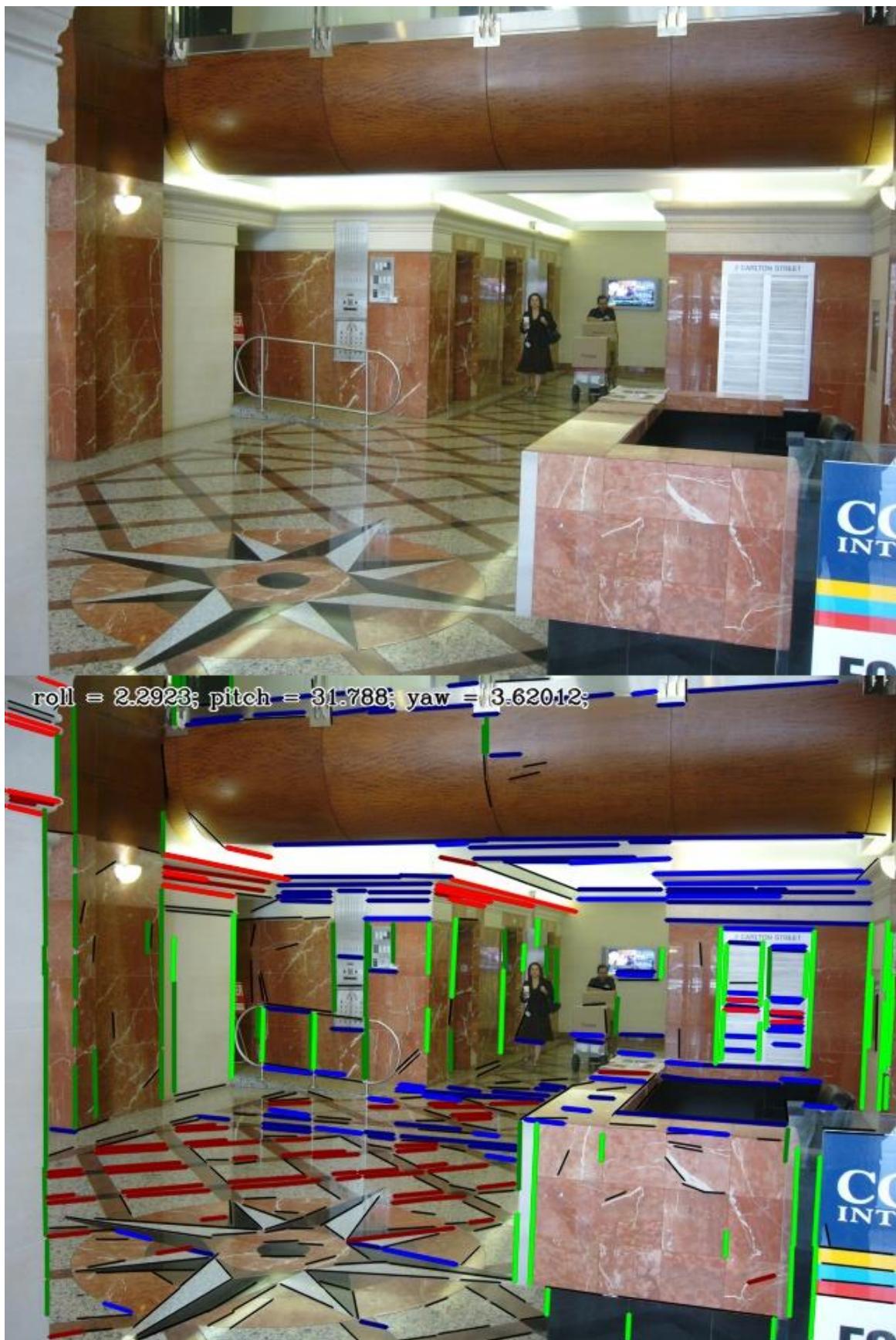


Рисунок 33 — Пример обработки изображения, на котором метод показал наихудшую точность среди набора изображений «внутри помещения». Средний вектор ошибки направлений ТСП в градусах — (14,74; 1,92; 14,90).

#### **4.4.2 Результаты тестирования метода на изображениях «вне помещения»**

В результате второго этапа тестирования было обработано 57 снимков городских сцен города Торонто из коллекции базы данных YorkUrbanDB. По аналогии с первым этапом тестирования для каждого из изображений было проведено по 20 запусков алгоритма.

Общее время обработки коллекции в среднем составило:

- $\approx 9,6$  сек или  $\approx 5,9$  fps — для запусков без ключа «`-noImage`», т.е. с формированием изображения–результата.
- $\approx 9,4$  сек или  $\approx 6,0$  fps — для запусков с ключом «`-noImage`».

Получены следующие результаты:

- $M_{MAX} = [6,34; 1,99; 7,07]$
- $M_{MIN} = [0,20; 0,26; 0,29]$
- $M_{AVG} = [1,81; 1,13; 1,94]$
- $V_{AVG} = [0,36; 0,10; 0,36]$

По итогам тестирования был построен график распределения ошибок. Получены следующие оценки по горизонтальным осям:

- 50 перцентиль  $\cong 1,7$  градуса;
- 75 перцентиль  $< 2,4$  градуса;
- 90 перцентиль  $< 3,7$  градуса.

И по вертикальной оси:

- 50 перцентиль  $\cong 1$  градус
- 75 перцентиль  $\cong 1,6$  градуса
- 90 перцентиль  $\cong 2$  градуса

Результаты распределения полученных направлений представлены на диаграммах с разделе дополнительных иллюстраций (Диаграмма 3, Диаграмма 4). На рис. 34–35 приведены примеры результатов обработки изображений, на которых метод показал наилучшую и наихудшую точность соответственно.

На наборе изображений городской среды метод показал себя тоже весьма хорошо. Средняя точность несколько ниже по сравнению результатами на наборе «внутри помещения». В большинстве случаев это связано с меньшим числом внутренних (для верных направлений ТСП) сегментов линий по отношению к общему их количеству. Также вне помещения чаще присутствуют:

- кластеры направлений, не являющихся ортогональными по отношению к остальным — не перпендикулярные друг другу улицы или дороги, углы зданий;
- объекты, порождающие хаотично направленные сегменты линий — деревья и кустарники;
- объекты, порождающие слегка отклоненные от верных направлений сегменты линий — люди и автомобили, трещины и линии разметки на дорогах, провисающие линии электропередач, накренившиеся столбы.

С другой стороны максимальная ошибка оказалась меньше, что скорее всего является особенностью самого набора, чем заслугой метода.

В целом, точность работы наряду со стабильностью очень высока, что позволяет предположить о состоятельности метода на «хороших» снимках, к которым можно отнести изображения из набора YorkUrbanDb. Под «хорошими» в данном случае понимаются изображения, на которых хорошо прослеживаются кластеры хотя бы двух направлений взаимно ортогональных ТСП.



roll = -15.6205; pitch = 32.7983; yaw = -7.39811;



Рисунок 34 — Пример обработки изображения, на котором метод показал наилучшую точность среди набора изображений «вне помещения». Средний вектор ошибки направлений ТСП в градусах — (0,20; 0,26; 0,29).

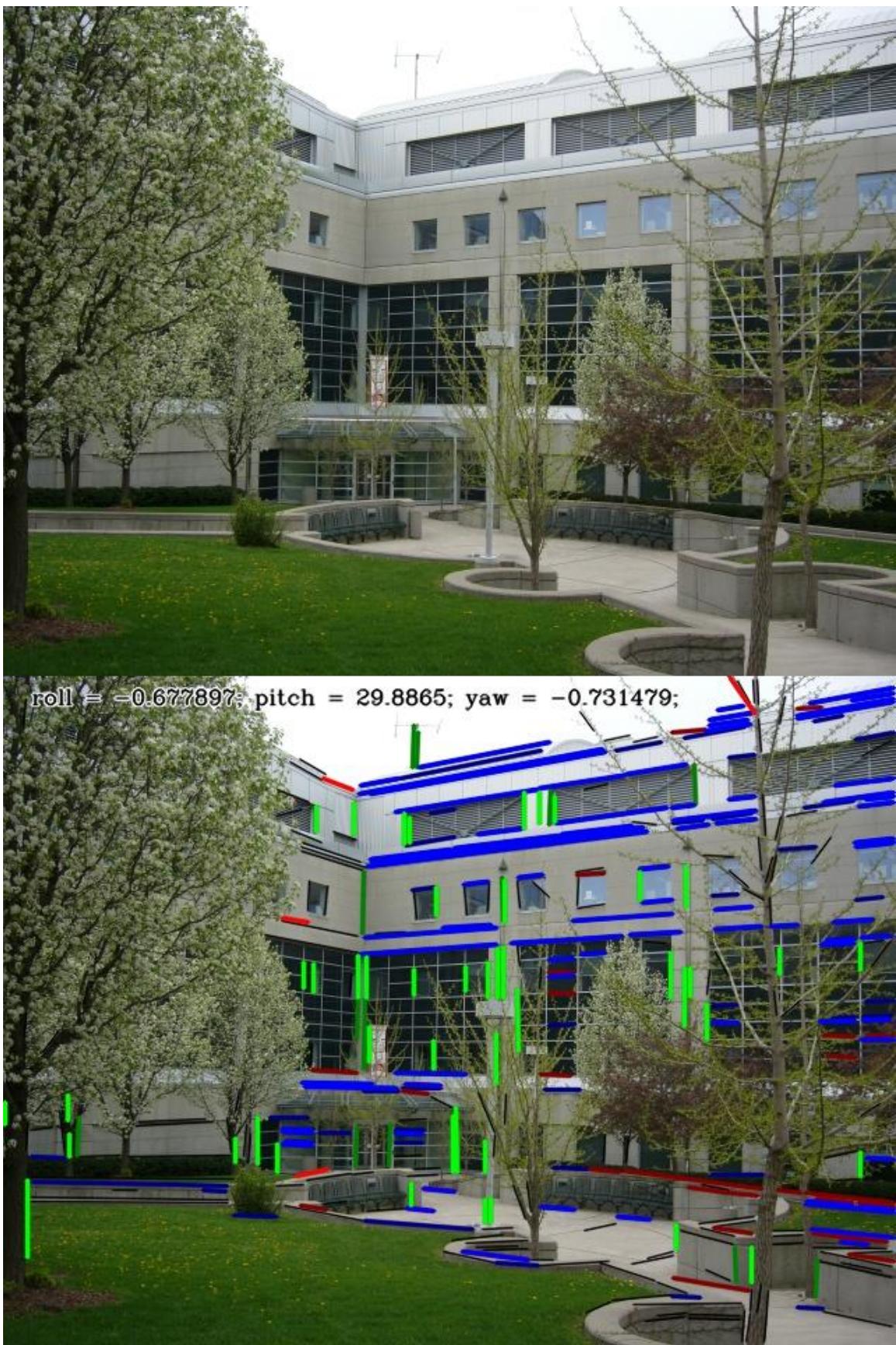


Рисунок 35 — Пример обработки изображения, на котором метод показал наихудшую точность среди набора изображений «вне помещения». Средний вектор ошибки направлений ТСП в градусах — (6,34; 1,99; 7,07).

#### 4.4.3 Результаты тестирования на случайных изображениях

Помимо тестирования работы метода на наборе изображений с откалиброванной камерой, был проведен ряд тестов на случайных изображениях, взятых из сети Интернет или сделанных вручную. Для данных изображений матрица калибровки была неизвестна.

На рис. 36 приведен пример работы метода на снимке, сделанном вручную внутри помещения. Точность оценки выборки кластеров в данном случае достаточно высока.

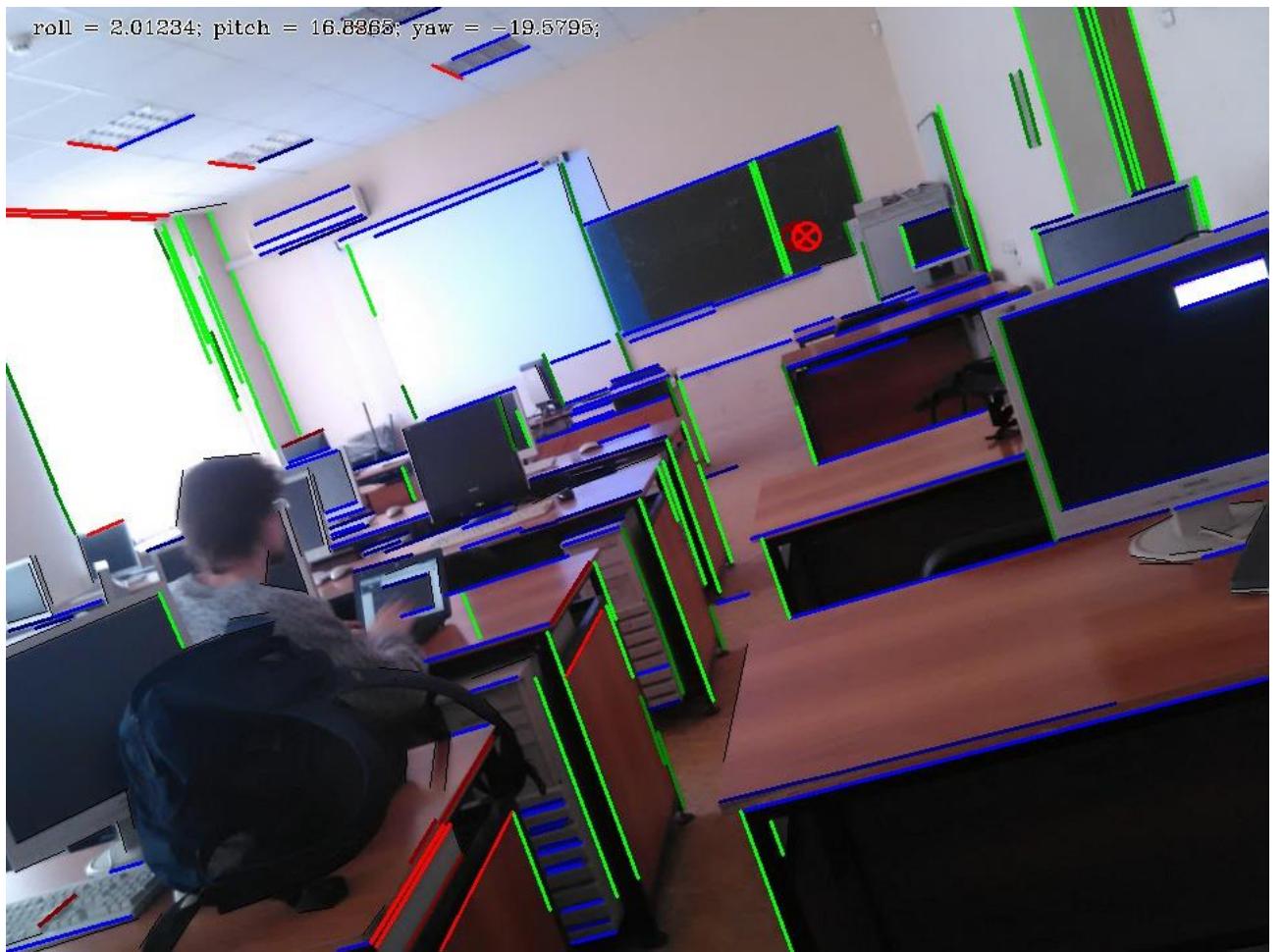


Рисунок 36 — Пример результата работы приложения на изображении внутри помещения, сделанном вручную с некалиброванной камеры.

На рис. 37 тестировалась обработка изображения картины, на которой художник сделал попытку соблюсти правила перспективы. Как видно, кластеризация прошла достаточно успешно, хотя значительное количество

выделенных линий перспективы отбракованы (выделены на рисунке черным цветом).



Рисунок 37 — Пример результата работы приложения на синтетическом тесте.

На рис. 38 приведен результат работы метода на изображении здания МИТ Stata, которое имеет неправильную форму. Результаты выделения кластеров и вычисления углов ориентации далеки от реальности. Как видно, на данном изображении вообще не прослеживаются взаимно ортогональные направления, что и послужило причиной неудачной работы метода.

Интересный случай показан на рис. 39 результата работы на синтетическом тесте. В случае красной и зеленой ТСП, алгоритм кластеризации выбрал не правильные пары сегментов. Это говорит о том, что в случае сильно ограниченного числа выделенных СЛ выбор кластеров может быть непредсказуем. Однако это скорее особенность синтетических тестов, так как на

реальных изображениях обычно достаточно большое число выделенных сегментов.



Рисунок 38 — Пример неудачного результата работы приложения на здании MIT Stata.

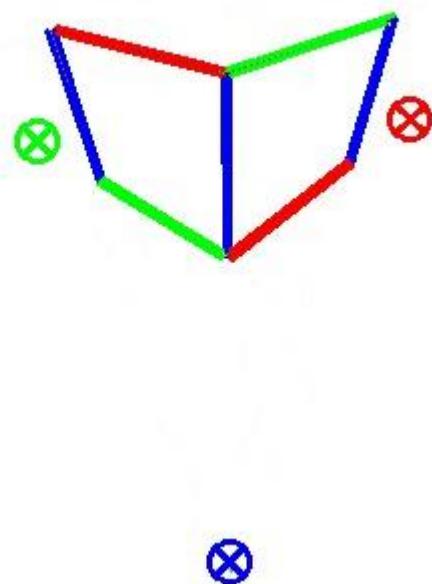


Рисунок 39 — Пример ошибки выделения ТСП в случае ограниченного числа СЛ.

При тестировании на изображениях живой природы в случае, когда заснятая местность не имела очертаний хотя бы двух взаимно ортогональных направлений, метод показал практически полную непригодность. Это в добавок усугублялось тем, что была неизвестна матрица калибровки.

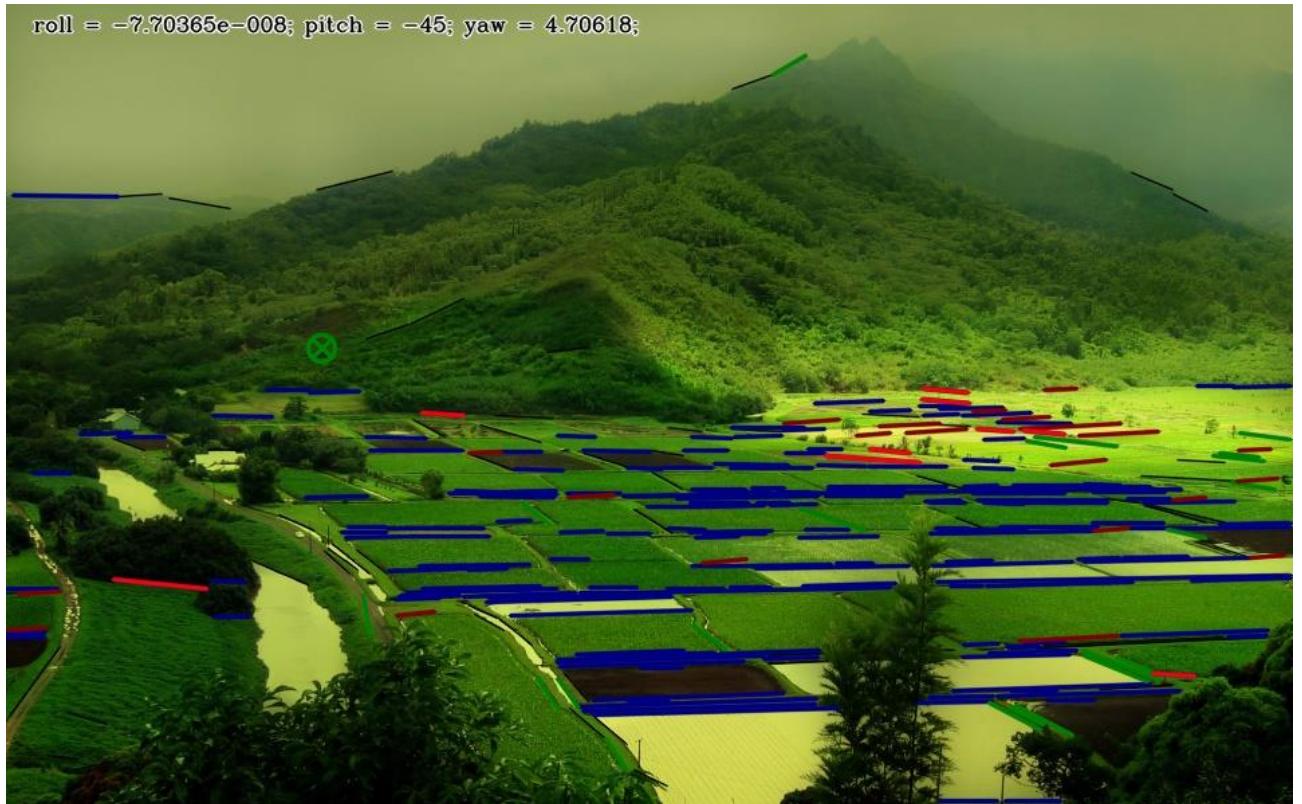


Рисунок 40 — Пример неудачной работы метода на изображении живой природы.

По итогам тестирования можно заключить, что для использования текущей реализации метода, при условии неизвестной матрицы калибровки, требуется усовершенствование алгоритма. Тоже самое относится к использованию приложения для определения углов ориентации по снимкам, на которых меньше двух взаимно ортогональных кластеров сегментов линий.

# **ГЛАВА 5. ОРГАНИЗАЦИОННО-ЭКОНОМИЧЕСКАЯ ЧАСТЬ**

Разрабатываемое в рамках дипломной работы программное обеспечение является одной из составных частей — гироскоп — системы навигации и позиционирования для автономной робототехники. Данный продукт позволяет определять пространственную ориентацию камеры, жестко прикрепленной к объекту позиционирования, на основе полученных с нее изображений. Разработанное ПО вычисляет по входным изображениям координаты точек схождения перспективы (ТСП), выводит отладочную информацию в виде изображения, полученного наложением на исходное изображение распознанных сегментов линий, соответствующих найденным ТСП, а также выводит углы Эйлера в нотации (1, 2, 3), т.е. углы крена, тангажа и рыскания.

На данный момент существует несколько вариантов реализаций систем позиционирования робототехники, в том числе аналогов, использующих методы компьютерного зрения. По сравнению глобальными (спутниковыми) системами навигации разработанное ПО позволяет получить более точные данные в условиях городской среды и внутри помещений, в частности. В отличие от инерциальных навигационных систем, наша не имеет дрейфа при сравнительных показателях точности. По сравнению же с аналогами, использующими методы SLAM и SfM, которые решают комплексную задачу позиционирования и навигации, разработанное приложение решает лишь задачу определения пространственной ориентации – сосредоточившись на одной задаче мы получаем выигрыш в производительности.

Целью данного раздела является расчет трудоемкости, продолжительности разработки программного обеспечения и сметы затрат.

## **5.1 Организация и планирование процесса разработки программы**

### **5.1.1 Техническое задание**

Для расчёта затрат на выполнение дипломного проекта используем техническое задание, представленное в таблице 12.

Таблица 12 — Техническое задание

| <b>№</b> | <b>Наименование</b>                             | <b>Значение</b>  |
|----------|---|--|
| 1        | Срок начала проекта                             | 10 марта 2015 г.   |
| 2        | Срок окончания проекта                          | 31 мая 2015 г.   |
| 3        | Количество листов А4 записки проекта            | 85   |
| 4        | Тип конечного носителя разрабатываемого проекта | Электронный информационный носитель, к которому прилагается описание проекта в виде отчёта |
| 5        | Планируемое число копий                         | Тиражирование данной продукции не планируется  |

### **5.1.2 Расчёт стоимости проекта**

Базовой экономической характеристикой для оценки дипломного проекта является её стоимость. Стоимость дипломного проекта, как экономическая категория, определяется по формуле [26].

$$C_T = C + P, \quad (40)$$

где

- $C_T$  — стоимость проектных работ,
- $C$  — себестоимость проектных работ,
- $P$  — прибыль.

Выразим прибыль через себестоимость работ в формуле (64):

$$P = R \times C, \quad (41)$$

где  $R$  – желаемый для исполнителя уровень рентабельности.

Исходя из формул (40) и (64), можем выразить стоимость проекта как показано в формуле (64):

$$C_T = C(1 + R) \quad (42)$$

Таким образом для расчёта стоимости проекта потребуется спрогнозировать себестоимость. Для её определения необходимо рассчитать каждую стадию сметы затрат [26]:

$$C = C_1 + C_2 + C_3 + C_4 + C_5, \quad (43)$$

где соответствующие  $C_i$  представлены в таблице 13.

Таблица 13 — Структура сметы затрат на выполнение проекта

| <b>№</b> | <b>Наименование статьи затрат</b> | <b>Затраты, %</b> |
|----------|-----------------------------------|-------------------|
| 1        | Материальные                      | 20                |

| <b>№</b> | <b>Наименование статьи затрат</b>           | <b>Затраты, %</b> |
|----------|---|-------------------|
| 2        | Заработка плата (основная и дополнительная) | 45                |
| 3        | Отчисления на социальные нужды              |                   |
| 4        | Амортизация оборудования                    | 20                |
| 5        | Прочие затраты                              | 15                |

Прочие затраты обычно составляют 15% [26] и включают в себя:

- оплату налогов;
- подготовку специальной научно-технической информации;
- проведение патентных исследований, научно-технических конкурсов и экспертиз;
- услуги всех видов связи;
- служебные командировки работников в Российской Федерации и за рубежом;
- расходы на сертификацию продукции;
- представительские расходы.

Практика использования таких данных и экономическая целесообразность показывает, что наилучший результат по точности прогноза получается, если в качестве искомой величины взять расчет затрат на заработную плату и отчисления на социальные нужды  $C_2 + C_3$  и через это значение определить себестоимость проекта. Исходя из значений доли статьи расходов к общей стоимости проекта и из (42), рассчитаем себестоимость проекта:

$$C = \frac{C_2 + C_3}{0.45} \quad (44)$$

### 5.1.3 Затраты на выплату исполнителям

В заработную плату включается основная и дополнительная заработка плата всех исполнителей, непосредственно занятых разработкой, с учетом их должностного оклада и времени участия в разработке. Дополнительную заработную плату в составе обобщённой определим следующим образом [26]:

$$C_2 = C_{21} + C_{22} = C_{21} + (C_{21} \times \alpha) = C_{21}(1 + \alpha), \quad (45)$$

где

- $C_2$  — заработная плата,
- $R$  — основная заработная плата,
- $C_{22}$  — дополнительная заработная плата,
- $\alpha$  — коэффициент отчислений на дополнительную заработную плату.

Коэффициент примем равным 100%, учитывая расходы на очередные отпуска, выплаты за выслугу лет и прочее. Таким образом, подставив это значение в формулу (44), получим значение заработной платы:

$$C_2 = 2 \times C_{21} \quad (46)$$

В настоящее время федеральным законом РФ №212-ФЗ от 24.07.2009 вместо единого социального налога определяются страховые взносы для отчисления в:

- пенсионный фонд РФ,
- фонд социального страхования,
- фонды обязательного медицинского страхования (федеральный и территориальный фонды).

Ставки страховых взносов в 2015 году для организаций, осуществляющих деятельность в области информационных технологий, за исключением организаций, заключивших с органами управления особыми экономическими зонами соглашения об осуществлении технико-внедренческой деятельности, [26] указаны в таблице 14.

Таблица 14 — Ставки страховых взносов на 2015 год

| <b>Получатель</b>                           | <b>Ставка страхового взноса</b> |
|---|---------------------------------|
| Пенсионный фонд РФ                          | 8%                              |
| Фонд социального страхования                | 2%                              |
| Фонд обязательного медицинского страхования | 4%                              |

Рассчитаем отчисления на социальные нужды с заработной платы:

$$C_3 = C_2 \times (H_{\Pi\Phi} + H_{\Phi\text{CC}} + H_{\Phi\text{OMC}}) = C_2 \times 0,14 \quad (47)$$

К основной заработной плате при выполнении проектных работ относится фонд оплаты труда (ФОТ) научных, инженерных и технических работников, рабочих научно-исследовательских и научно-технических отделов, принимающих непосредственное участие в НИР. ФОТ работников за выполнение разработки определяется следующим образом:

(48)

$$C_{21} = \sum_i R_i \times T_i \times N_i,$$

где

- $R_i$  – среднедневный ФОТ рабочих  $i$ -й специальности,
- $T_i$  – количество дней работ для рабочих  $i$ -й специальности,
- $N_i$  – количество рабочих  $i$ -й специальности.

Для расчёта количества рабочих дней и количества рабочих потребуется рассчитать трудоёмкость проекта. Разработка программного продукта состоит из пяти основных этапов, состав работ которых указан в таблице 15 [26]:

- техническое задание;
- эскизный проект;
- технический проект;
- рабочий проект;
- внедрение.

Таблица 15 — Этапы разработки программного продукта

| <b>№ этапа</b> | <b>Название этапа</b>       | <b>Общий состав работ этапа</b>   |
|----------------|-----------------------------|---|
| 1              | Техническое задание<br>(ТЗ) | Разработка ТЗ.  |
| 2              | Эскизный проект (ЭП)        | Исследование существующего программного продукта.<br>Уточнение структуры и формы представления входных и выходных данных.<br>Разработка алгоритма решения задачи. |

| <b>№ этапа</b> | <b>Название этапа</b>   | <b>Общий состав работ этапа</b>   |
|----------------|-------------------------|---|
|                |                         | Разработка структуры программы.<br>Разработка пояснительной записи.<br>Согласование и утверждение технического проекта.   |
| 3              | Технический проект (ТП) | Разработка алгоритмов (общих алгоритмов и структуры данных, структуры основных и вспомогательных модулей и др.)   |
| 4              | Рабочий проект (РП)     | Описание программы на языке программирования.<br>Разработка, создание и утверждение порядка и методики испытаний, корректировка программы.  |
| 5              | Внедрение (В)           | Разработка программной документации.<br>Подготовка и передача программы и программной документации для сопровождения и изготовления, оформления и утверждения акта о передаче ПП на сопровождение. Передача ПП заказчику. |

Исходя из расчёта трудоёмкости всего проекта, определим трудоёмкость каждого этапа и по заданным срокам проекта требуемое количество работников.

Вначале рассчитаем трудоёмкость проекта по нормативно-статистическому методу [26]. За единицу нормирования принимается разработка одного листа технической документации формата А4 эскизного проекта.

$$T_{\Pi} = \frac{L \times T_{\text{HB}} \times K_{\text{ови}} \times K_{\text{ck}} \times K_{\text{бо}} \times K_{\text{тр}} \times K_{\text{аяп}}}{\mu}, \quad (49)$$

где

- $T_{\Pi}$  — трудоёмкость всего проекта, чел/час;
- $L$  — количество требуемых листов документации, шт;
- $T_{\text{hb}}$  — норма времени на разработку одного листа формата А4, час/шт;
- $K_{\text{ови}}$  — коэффициент, учитывающий объём входной информации;
- $K_{\text{ck}}$  — коэффициент, учитывающий сложность контроля информации;
- $K_{\text{бо}}$  — коэффициент, учитывающий вид обработки информации (режим обработки информации);
- $K_{\text{тр}}$  — поправочный коэффициент по степени применения типовых проектных решений, пакетов прикладных программ, типовых проектов, типовых программ и стандартных модулей;
- $K_{\text{аяп}}$  — коэффициент учёта уровня алгоритмического языка программирования;
- $\mu$  — доля трудозатрат в общем проекте.

Коэффициент, учитывающий объём входной информации зависит от количества наборов входных:

$$K_{\text{ови}} = \frac{K_{\Pi} n_{\Pi} + K_{\text{HC}} n_{\text{HC}} + K_{\text{Б}} n_{\text{Б}}}{n_{\Pi} + n_{\text{HC}} + n_{\text{Б}}}, \quad (50)$$

где

- $K_{\Pi}$ ,  $K_{\text{HC}}$  и  $K_{\text{Б}}$  – значения коэффициентов учета вида используемой

информации для переменной, нормативно-справочной информации и баз данных соответственно;

- $n_{\Pi}$ ,  $n_{HC}$  и  $n_B$  – количество наборов данных переменной, нормативно-справочной информации и базы данных соответственно.

На вход программного продукта должна подаваться информация одного вида — файл с изображением, на котором следует обнаружить точки схождения перспективы.

На выходе алгоритма два вида информации:

- файл с сохраненным изображением, составленным из, наложенных на оригинальное изображение групп сегментов линий, каждая из которых выделена своим цветом и соответствует одной из найденных точек схождения перспективы.
- текстовый файл с сохраненной матрицей векторов направлений, соответствующих найденным точкам схождения перспективы.

По степени новизны программной продукт может быть отнесён к одной из четырех групп, представленный в таблице 16. А по степени сложности алгоритма к одной из трёх групп, указанных в таблице 17.

В данном случае программа относится к группе «В», поскольку существуют программные комплексы, реализующие аналогичный функционал. По степени сложности программа относится к группе 1.

Для группы новизны «В» и сложности алгоритма группы 1 значения коэффициентов равны [26]:  $K_{\Pi} = 1,2$ ;  $K_{HC} = 0,65$ ;  $K_B = 0,54$ . Подставив их в (64), получим:

$$K_{ovi} = \frac{1,2 \cdot 1 + 0,65 \cdot 0 + 0,54 \cdot 0}{1} = 1,2 \quad (51)$$

Таблица 16 — Классификация степени новизны разрабатываемого программного продукта

| <b>Название группы</b> | <b>Описание</b>  |
|------------------------|--|
| A                      | Разработка программных комплексов, требующих использования принципиально новых методов их создания, проведение НИРС и т.п. |
| Б                      | Разработка программной продукции, не имеющей аналогов, в том числе разработка пакетов прикладных программ.                 |
| В                      | Разработка программной продукции, имеющей аналоги.   |
| Г                      | Разработка программной продукции, основанной на привязке типовых проектных решений.  |

Таблица 17 — Классификация степени сложности алгоритма программной продукции

| <b>Степень сложности</b> | <b>Описание</b>  |
|--------------------------|--|
| 1                        | Программная продукция, реализующая оптимизационные и моделирующие алгоритмы    |
| 2                        | Программная продукция, реализующая учётно-статистические алгоритмы             |
| 3                        | Программная продукция, реализующая алгоритмы стандартных методов решения задач |

Определим требуемые в (49) коэффициенты. Количество требуемых листов документации исходя из технического задания равно  $L = 85$ . Норма времени на разработку документации одного листа формата А4 равно  $T_{\text{нв}} = 2$  часам [26]. Согласно таблицам из [26] коэффициент учёта режима обработки

информации  $K_{\text{бо}}$  для технического проекта с обработкой информации в реальном времени группы новизны «В» равен  $K_{\text{бо}} = 1,26$ . Коэффициент, учитывающий сложность контроля информации, для данной специфики задачи равен  $K_{\text{ск}} = 1,16$ .

Данный программный продукт использует реализации части алгоритмов из библиотеки компьютерного зрения opencv, а также в качестве платформы разработки, обеспечивающей набором примитивов и методов для работы с изображениями, геометрией и так далее. Оценим поправочный коэффициент по степени применения типовых проектных решений, пакетов прикладных программ равным  $K_{\text{тр}} = 0,6$ . Программный код разрабатывается на языке высокого уровня, поэтому коэффициент учёта уровня алгоритмического языка программирования равен  $K_{\text{аяп}} = 1,0$ .

Таким образом, подставив коэффициенты в (49), получим трудоёмкость рабочего проекта:

$$T_{\text{п}} = \frac{85 \times 2 \times 1,2 \times 1,16 \times 1,26 \times 0,6 \times 1,0}{20\%} \approx 894,5 \quad (52)$$

Для того, чтобы определить количество человек, требуемых для выполнения каждого из этапов разработки, необходимо определить трудоёмкость каждого этапа. В таблице 18 представлены доли трудоёмкости каждого этапа, согласно [26], и рассчитанное по формуле абсолютное его значение:

$$T_i = \rho \times T_{\text{п}}, \quad (53)$$

где  $\rho$  — доля каждого этапа.

Таблица 18 — Трудоёмкость этапов разработки программного продукта

| №<br>этапа | Название этапа      | Трудоёмкость |         |
|------------|---------------------|--------------|---------|
|            |                     | %            | чел/час |
| 1          | Техническое задание | 10           | 89,5    |
| 2          | Эскизный проект     | 15           | 134,3   |
| 3          | Технический проект  | 30           | 268,5   |
| 4          | Рабочий проект      | 35           | 313,2   |
| 5          | Внедрение           | 10           | 89,5    |
|            | Всего               | 100          | 895     |

При выполнении разработки требуемое количество исполнителей для выполнения этапа в заданный срок равен:

$$R_{ti} = \frac{t_i K_d}{F_n K_h}, \quad (54)$$

где

- $t_i$  — трудоёмкость этапа, чел/час;
- $K_d$  — коэффициент дополнительных работ, учитывающий затраты времени на работы, не предусмотренные нормативами,  $K_d = 1,15$ ;
- $F_n$  — фонд рабочего времени исполнителя за период, определяемый сроками;
- $K_h$  — коэффициент, учитывающий выполнение норм,  $K_h = 1,15$ .

Фонд рабочего времени каждого исполнителя за период с 10 марта 2015 года по 31 мая 2015 года рассчитывается следующим образом:

$$F_{\Pi} = T * F_M, \quad (55)$$

где

- $T$  — время выполнения проекта в месяцах (устанавливается в ТЗ и для этого проекта равно 2,6 месяца),
- $F_M$  — фонд времени в текущем месяце, который рассчитывается из учета общего числа дней в году, числа выходных и праздничных дней, рассчитываемого так:

$$F_M = \frac{t_p * (D_K - (D_B + D_{\Pi}))}{12}, \quad (56)$$

где

- $t_p$  — продолжительность рабочего дня,
- $D_K$  — общее число дней в году,
- $D_B$  — число выходных дней в году,
- $D_{\Pi}$  — число праздничных дней в году.

Нерабочие праздничные дни в году устанавливается соответствующим законом. Подставив соответствующие значения, рассчитаем среднемесячный фонд времени:

$$F_M = \frac{8 * (365 - 118)}{12} = 164,6 \frac{\text{ч}}{\text{мес}} \quad (57)$$

Таким образом, исходя из (55), фонд времени на период работы, определённых в техническом задании, равен 428,1 часам.

#### **5.1.4 Определение количества исполнителей**

При равномерном распределении работ для выполнения проекта в срок, установленный техническим заданием — 31 мая 2015 года, среднее значение требуемого количества исполнителей равно:

$$R_{ti} = \frac{894,5 \cdot 1,15}{428,1 \cdot 1,15} = 2,09 \text{ чел} \quad (58)$$

#### **5.1.5 Календарный график выполнения работ**

Для целей планирования и контроля работ проекта можно применить календарный ленточный график (диаграмма Ганта) — на оси X показывают календарные дни (по рабочим неделям) от начала проекта до его завершения, а по оси Y — выполняемые этапы работ.

Продолжительность выполнения работ без учёта выходных и праздничных дней по определяется:

$$T_i = \frac{t_i + Q}{n_i}, \quad (59)$$

где

- $t_i$  — трудоёмкость  $i$ -й работы, чел.-часы;
- $Q$  — трудоёмкость дополнительных работ для исполнителя, чел.-часы;
- $n_i$  — количество исполнителей для  $i$ -й работы.

В результате была построена диаграмма Ганта указанных работ с учётом праздничных и выходных дней, которая представлена в таблице 22 в приложении 1.

Длительность стадий проектных равна:

$$L_i = \frac{t_i(1 + p)}{P_i g}, \quad (60)$$

где

- $L_i$  — продолжительность стадии, рабочие дни;
- $t_i$  — трудоёмкость этапа, чел/час;
- $p = 0,1\dots0,3$  — доля дополнительных работ;
- $g$  — учитываемая продолжительность рабочего дня, часов/рабочий день,  $g = 8$ .

Необходимо распределить этапы работ для целого числа исполнителей с таким расчётом, чтобы уложиться в требуемые сроки. Такое распределение приведено в таблице 19.

Таблица 19 — Оптимизированное количество исполнителей проектных работ

| <b>№ стадии</b>           | <b>1</b> | <b>2</b> | <b>3</b> | <b>4</b> | <b>5</b> |
|---------------------------|----------|----------|----------|----------|----------|
| Трудоёмкость, чел/час     | 89,5     | 134,3    | 268,5    | 313,2    | 89,5     |
| Доля дополнительных работ | 0,1      | 0,1      | 0,3      | 0,3      | 0,2      |
| Количество исполнителей   | 2        | 2        | 2        | 2        | 2        |

В соответствии с Единым Тарифно–Квалификационным справочником квалификация работников, выполняющих этапы работ [36], назначается

следующим образом:

- инженер–программист 1 категории,
- инженер по научно–технической информации 1 категории.

Средняя численность состава исполнителей при реализации проекта разработки и внедрения ПО:

$$N = \frac{\tau_{\text{ПП}}}{F} \quad (61)$$

где

- $\tau_{\text{ПП}}$  — затраты труда на выполнение проекта (разработка и внедрение ПО),
- $F$  — фонд рабочего времени.

## 5.2 Расчёт сметы затрат

Затраты на выполнение проекта состоят из прямых затрат (заработка исполнителям, затраты на закупку или аренду оборудования, затраты на организацию рабочих мест), и косвенных затрат (т.н. накладные расходы) вычисляются по формуле:

$$K = C_{\text{ЗАРП}} + C_{\text{ОБ}} + C_{\text{ОРГ}} + C_{\text{НАКЛ}} \quad (62)$$

где

- $C_{\text{ЗАРП}}$  — заработка исполнителей;
- $C_{\text{ОБ}}$  — затраты на обеспечение необходимым оборудованием;
- $C_{\text{ОРГ}}$  — затраты на организацию рабочих мест;

–  $C_{\text{НАКЛ}}$  — накладные расходы.

В нашем проекте разработчиком ПО является программист С++, а проектировщиком — системный аналитик. Средние заработные платы по Москве были определены с помощью регулярных пресс-релизов электронного ресурса superjob.ru [27, 28]. Для программиста С++ (диапазон 3 в градации пресс-релиза) составляют 101500 рублей в месяц при полной рабочей неделе [27], а аналитика (диапазон 3) — 105000 рублей в месяц [28].

Итоговые заработные платы для работников указаны в таблице 20.

Следовательно, общие затраты на заработную плату исполнителям проекта составят  $C_{\text{з.осн.}} = 497682,9$  руб. Расходы на дополнительную заработную плату учитывают все выплаты непосредственным исполнителям за время, не проработанное на производстве, но предусмотренное законодательством. Величина этих выплат составляет 20% от размера основной заработной платы и вычисляется по формуле:

$$C_{\text{з.доп.}} = 0,2 * C_{\text{з.осн.}} \quad (63)$$

Таблица 20 — Затраты на основную заработную плату сотрудников

| №     | Должность     | «Чистый» оклад, руб. | Почасовой оклад, руб. | Трудозатраты, чел.-час | Затраты на зарплату, руб. |
|-------|---------------|----------------------|-----------------------|------------------------|---------------------------|
| 1     | Проектировщик | 105 000              | 638                   | 290,7                  | 185 466,6                 |
| 2     | Разработчик   | 101 500              | 517                   | 603,9                  | 312 216,3                 |
| Итого |               |                      |                       |                        | 497 682,9                 |

Рассчитаем расходы на дополнительную заработную:

$$C_{3.\text{доп.}} = 0,2 * 497682,9 = 99536,58\text{руб.} \quad (35)$$

В настоящее время федеральным законом РФ №212-ФЗ от 24.07.2009 вместо единого социального налога определяются страховые взносы для отчисления в:

- пенсионный фонд РФ,
- фонд социального страхования,
- фонды обязательного медицинского страхования (федеральный и территориальный фонды).

Ставки страховых взносов в 2014 году для организаций, осуществляющих деятельность в области информационных технологий, за исключением организаций, заключивших с органами управления особыми экономическими зонами соглашения об осуществлении технико-внедренческой деятельности указаны в таблице 21.

Таблица 21 — Ставки страховых взносов на 2014 год

| <b>Получатель</b>                           | <b>Ставка страхового взноса</b> |
|---|---------------------------------|
| Пенсионный фонд РФ                          | 8%                              |
| Фонд социального страхования                | 2%                              |
| Фонд обязательного медицинского страхования | 4%                              |

Рассчитаем отчисления с заработной платы:

$$C_{3.\text{отч.}} = (C_{3.\text{осн.}} + C_{3.\text{доп.}}) * (H_{\Pi\Phi} + H_{\Phi\text{СС}} + H_{\Phi\text{ФОМС}}) = 83610,7\text{руб.}$$

Таким образом, получим общие затраты на заработную плату:

$$C_{\text{ЗАРП}} = 497682,9 + 99536,6 + 83610,7 = 680830,2 \text{ руб.}$$

### 5.2.1 Суммарные затраты

Суммарные затраты вычисляются как сумма всех:

$$K = C_{\text{ЗАРП}} + C_{\text{ОБ}} + C_{\text{ОРГ}} + C_{\text{НАКЛ}} \quad (64)$$

Определим затраты на реализацию проекта:

$$K = 1512956 \text{ руб.}$$

Таким образом для реализации данного проекта необходимы программист C++ (трудозатраты составят 604 часа) и системный аналитик (трудозатраты составят 291 час). Продолжительность выполнения проекта составляет 67 дней с учётом выходных и праздничных дней.

Расходы на разработку продукта равны 1512956 рублей.

## ЗАКЛЮЧЕНИЕ

В представленной работе проведен анализ подходов к ориентации автономного движущегося объекта, в том числе основанных на методах компьютерного зрения.

Апробирован метод нахождения углов пространственной ориентации моноокулярной камеры с использованием точек схождения перспективы (метод Хуттунена–Пише) [1].

Разработано прикладное программное обеспечение, которое состоит из следующих функциональных блоков:

- организация конфигурации запуска приложения и управления тестированием;
- управление процессом обработки изображения;
- управление процессом тестирования набора изображений;

Решены вопросы технологического характера, связанные с использованием открытой библиотеки алгоритмов компьютерного зрения OpenCV [23], а также с конвертацией данных.

Тестирования проводилось на двух наборах изображений.

На наборе изображений YorkUrbanDb [2] Йоркского университета, состоящего из пейзажей города Торонто, а также внутренних помещений.

В результате тестирования было обработано 45 снимков внутри помещений из коллекции базы данных YorkUrbanDB. Среднее время обработки всех снимков составило  $\approx 4,28$  сек, что эквивалентно  $\approx 10,5$  fps. Кроме того, было обработано 57 снимков городских сцен города Торонто из коллекции базы данных YorkUrbanDB. Среднее время обработки всех снимков составило  $\approx 9,4$  сек, что эквивалентно  $\approx 6,0$  fps.

Были получены диаграммы, иллюстрирующие точность подхода.

На обоих типах изображений метод показал достаточно высокий уровень точности и стабильности. В 90% случаев погрешность метода не превышает 4 градусов по каждой из осей, в 75% — 2,5 градуса. Средняя точность не превышает 2 градусов, что сравнимо с точностью ИНС потребительского класса. Средняя среднеквадратичного отклонения погрешности не превышает 0,4 градусов по каждой из осей.

Второй набор данных предназначался для оценки погрешностей при регистрации изображений, снятых некалиброванной камерой, а также синтетических изображений (картин). В этом случае наблюдалось значительное количество «отбракованных» линий схождения перспективы, хотя «принятые» линии кластеризовались верно.

Реализацию данного метода можно рассматривать перспективным решением в качестве подсистемы системы навигации и позиционирования автономной самоуправляемой техники.

По проведенному экономическому исследованию была рассчитана величина затрат на реализацию проекта, которая составила 1512956 рублей.

В качестве перспектив представленной работы можно рассматривать развитие и применение метода в условиях природного и городского окружения без наличия четко выраженных взаимно ортогональных направлений, а также модификаций, повышающих точность такого позиционирования. Кроме того, планируется оценить точность применения метода по серии снимков.

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Хуттунен В., Пише Р. Гирокоп на основе монокулярной камеры // Гирокопия и навигация, 2012. - №2(77). – стр. 69–81.
2. The York Urban Line Segment Database // Elder Laboratory: Human & Computer Vision, York University. — Электронный ресурс — Режим доступа: <http://www.elderlab.yorku.ca/YorkUrbanDB/>
3. Технологии позиционирования в реальном времени. — Электронный ресурс — Режим доступа: <http://www.rtlsnet.ru/technology/view/4>
4. О навигации // Сайт Информационно-аналитического центра координатно-временного и навигационного обеспечения ФГУП ЦНИИмаш. — Электронный ресурс — Режим доступа: <https://www.glonass-iac.ru/guide/navfaq.php>
5. Velodyne LiDAR // IDTechEx. — Электронный ресурс — Режим доступа: <http://portal.idtechex.com/popup/company-profile.asp?companyprofileid=555>
6. Blanco J. L. One way Google's cars localize. — Электронный ресурс — Режим доступа: <http://mappingignorance.org/2014/04/07/one-way-googles-cars-localize/>
7. Riisgaard S., Blas M. R.. SLAM For Dummies (A Tutorial Approach to Simultaneous Localization and Mapping) // MIT OpenCourseWare, Massachusetts Institute of Technology. — Электронный ресурс — Режим доступа: [http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam blas\\_repo.pdf](http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam blas_repo.pdf)
8. Grompone von Gioi Rafael [и др.]. LSD: a Line Segment Detector // Image Processing On Line, 2012 . – стр. 35–55 : Т. 2.
9. Fischler Martin A., Bolles Robert C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated

- cartography // Communications of the ACM. – NY: ACM New York, USA, 1981 – 6. – стр. 381–395 : Т. 24.
10. Wall Michael E., Rechtsteiner Andreas, Rocha Luis M. Singular value decomposition and principal component analysis // A Practical Approach to Microarray Data Analysis. - Norwell: Kluwer, 2003. - стр. 91–109.
11. Montiel J., Zisserman A. Automated Architectural Acquisition from a Camera Undergoing Planar Motion // International Symposium on Virtual and Augmented Architecture. - Dublin, Ireland: [б.н.], 2001. - стр. 207–218.
12. Cipolla R., Robertson D., Boyer E. PhotoBuilder — 3D models of architectural scenes from uncalibrated images // Multimedia Computing and Systems, 1999. IEEE International Conference. - [б.м.] : IEEE, 1999. - Т. 1.
13. Caprile B., Torre V. Using vanishing points for camera calibration // International Journal of Computer Vision. - [б.м.]: Kluwer Academic Publishers Hingham, MA, USA, 1990 г. - 2, стр. 127–140 : Т. 4.
14. Cipolla R., Drummond T., Robertson D. Camera Calibration From Vanishing Points in Images of Architectural Scenes // British Machine Vision Conference. - 1999.
15. He B.W., Li Y.F. Camera calibration from vanishing points in a vision system // Optics and Laser Technology. - 2008 . - стр. 555–561.
16. Kosecka J., Zhang W. Video Compass // ECCV '02 Proceedings of the 7th European Conference on Computer Vision — Part IV. - [б.м.] : Springer-Verlag, London, UK, 2002. - Т. стр. 476–490.
17. Tardif J.-P. Non-iterative Approach for Fast and Accurate Vanishing Point Detection // 12th IEEE International Conference on Computer Vision. - [б.м.] : IEEE, Kyoto, Japan, 2009.
18. Canny J. A Computational Approach To Edge Detection // Pattern Analysis and Machine Intelligence. - [б.м.] : IEEE, 1986. - 6, стр. 679–698 : Т. 8.

19. Coughlan J. M., Yuille A. L. Manhattan world: Compass direction from a single image by Bayesian inference // Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference. [б.м.] : IEEE, Kerkyra, 1999. – стр. 941–947.
20. Denis P., Elder J. H., Estrada F. J. Efficient Edge-Based Methods for Estimating Manhattan Frames in Urban Imagery // European Conference on Computer Vision. - 2008. - T. стр. 197–210.
21. Третьяк Е. [и др.] Geometric image parsing in man-made environments // International Journal of Computer Vision. - [б.м.] : Kluwer Academic Publishers: Hingham, MA, USA, 2011. – стр. 305–321.
22. Toldo R., Fusiello A. Robust Multiple Structures Estimation with J-Linkage // Computer Vision – ECCV 2008 // Lecture Notes in Computer Science. – 2008. – стр. 537–547
23. OpenCV (Open Source Computer Vision). — Электронный ресурс — Режим доступа: <http://opencv.org/>
24. CMake — cross-platform free and open-source software for managing the build process of software. — Электронный ресурс — Режим доступа: <http://www.cmake.org/>
25. OpenCV 3.0.0. Installation in Windows // OpenCV 3.0.0-dev documentation. — Электронный ресурс — Режим доступа: [http://docs.opencv.org/3.0-beta/doc/tutorials/introduction/windows\\_install/windows\\_install.html](http://docs.opencv.org/3.0-beta/doc/tutorials/introduction/windows_install/windows_install.html)
26. Сажин Ю.Б. Самохин С.В. Выполнение организационно-экономической части дипломного проекта по разработке и использованию программного продукта: Учебно-методическое пособие. – М. : Изд-во МГТУ им. Баумана, 2006. – 60 с.
27. Superjob.ru: средняя зарплата программиста C++ // Superjob.ru, Пресс-релиз. — Электронный ресурс — Режим доступа:

<http://www.it-analytics.ru/analytics/trends/71057.html>

28. Средняя зарплата системного аналитика // Superjob.ru, Пресс-релиз. —

Электронный ресурс — Режим доступа:

<http://www.it-analytics.ru/analytics/trends/72773.html>

## ДОПОЛНИТЕЛЬНЫЕ ИЛЛЮСТРАЦИИ

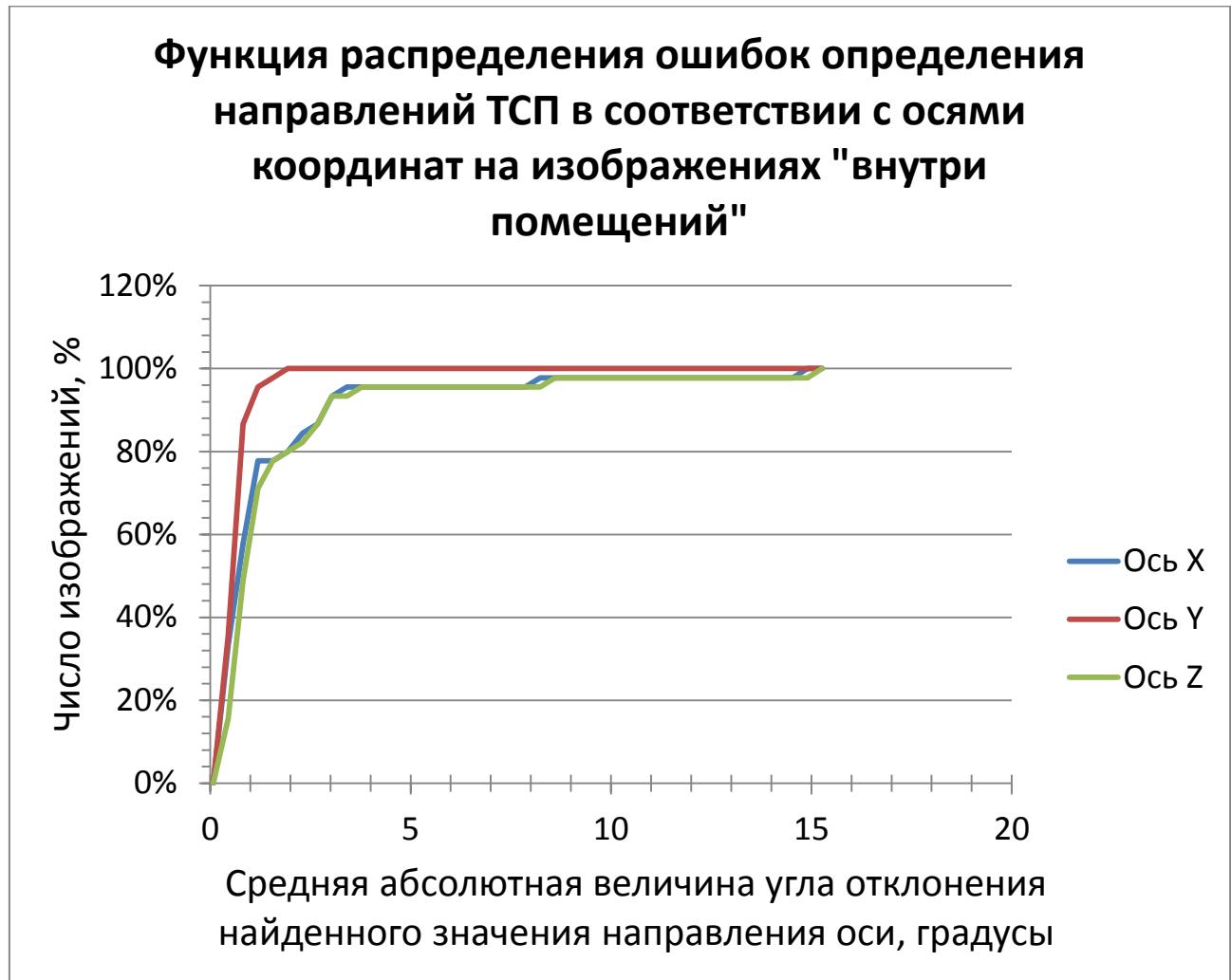


Диаграмма 1 — Диаграмма функции распределения точности найденных направлений осей координат на изображениях «внутри помещения».

**Плотность распределения ошибок определения  
направлений ТСП в соответствии с осями  
координат на изображениях "внутри  
помещений"**

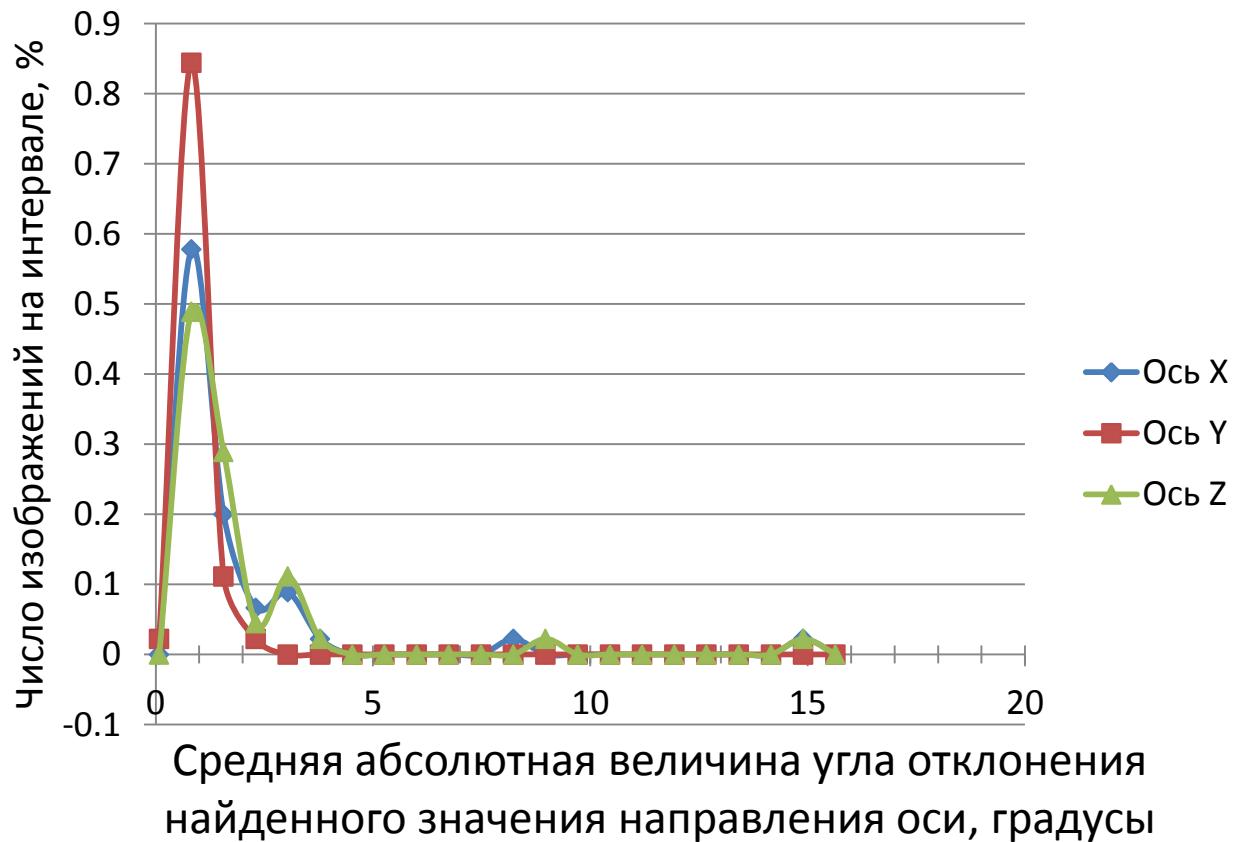


Диаграмма 2 — Диаграмма плотности распределения точности найденных направлений осей координат на изображениях «внутри помещения».

**Функция распределения ошибок определения  
направлений ТСП в соответствии с осями  
координат на изображениях "вне помещений"**

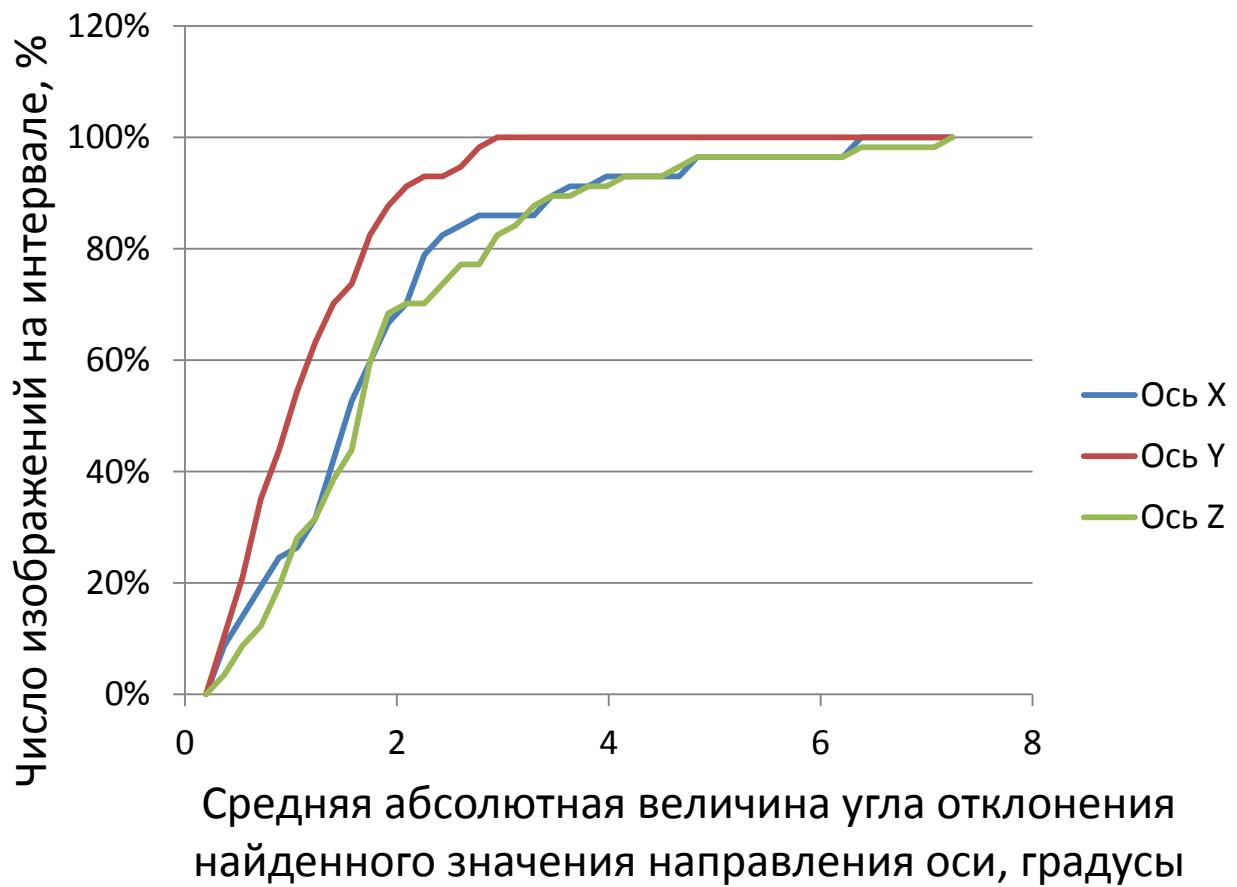


Диаграмма 3 — Диаграмма функции распределения точности найденных направлений осей координат на изображениях «вне помещения».

**Плотность распределения ошибок определения  
направлений ТСП в соответствии с осями  
координат на изображениях "вне помещений"**

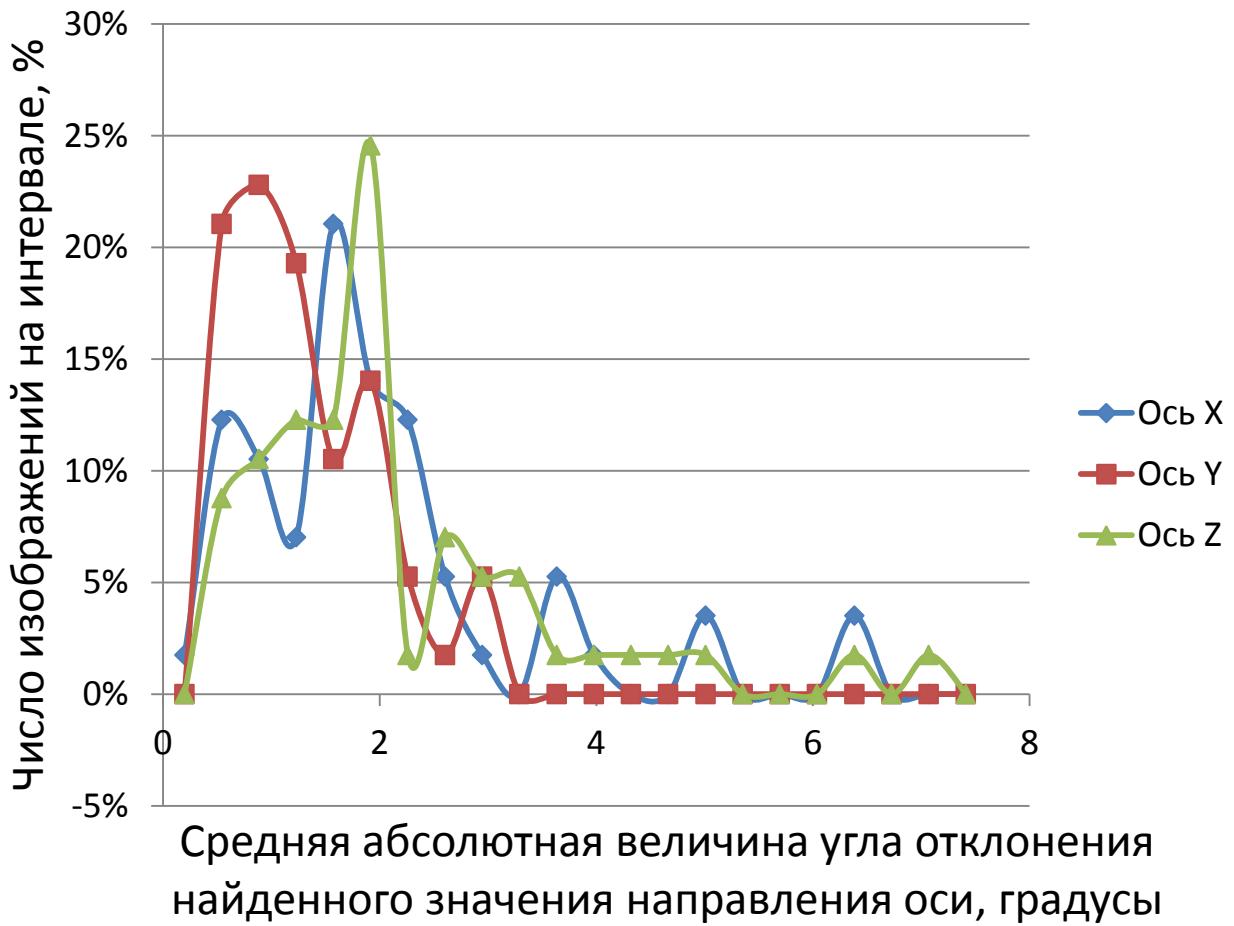


Диаграмма 4 — Диаграмма плотности распределения точности найденных направлений осей координат на изображениях «вне помещения».

## ПРИЛОЖЕНИЕ 1. ДИАГРАММА ГАНТА ВЫПОЛНЯЕМЫХ РАБОТ

Таблица 22 — Диаграмма Ганта выполняемых работ

| № | Название этапа      | Продолжительность, раб. дни | Исполнители                              |   | Число | Календарные дни         |             |             |             |             |             |             |             |             |             |             |             |  |
|---|---------------------|-----------------------------|--|---|-------|-------------------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|--|
|   |                     |                             | Категория                                |   |       | Количество рабочих дней |             |             |             |             |             |             |             |             |             |             |             |  |
|   |                     |                             |  |   |       | 10.03-15.03             | 16.03-22.03 | 23.03-29.03 | 30.03-05.04 | 06.04-12.04 | 13.04-19.04 | 20.04-26.04 | 27.04-03.05 | 04.05-10.05 | 11.05-17.05 | 18.05-24.05 | 25.05-31.05 |  |
| 1 | Техническое задание | 6                           | инженер-программист 1 категории          | 1 |       | 4                       | 2           |             |             |             |             |             |             |             |             |             |             |  |
|   |                     |                             | инженер по научно-технической информации | 1 |       |                         |             |             |             |             |             |             |             |             |             |             |             |  |
| 2 | Эскизный проект     | 8                           | инженер-программист 1 категории          | 1 |       |                         | 3           | 5           |             |             |             |             |             |             |             |             |             |  |
|   |                     |                             | инженер по научно-технической информации | 1 |       |                         |             |             |             |             |             |             |             |             |             |             |             |  |
| 3 | Технический проект  | 17                          | инженер-программист 1 категории          | 1 |       |                         |             |             | 5           | 5           | 5           | 2           |             |             |             |             |             |  |
|   |                     |                             | инженер по научно-технической информации | 1 |       |                         |             |             |             |             |             |             |             |             |             |             |             |  |
| 4 | Рабочий проект      | 19                          | инженер-программист 1 категории          | 2 |       |                         |             |             |             |             |             |             | 3           | 4           | 4           | 4           | 4           |  |
| 5 | Внедрение           | 6                           | инженер-программист 1 категории          | 1 |       |                         |             |             |             |             |             |             |             |             | 1           | 5           |             |  |
|   |                     |                             | инженер по научно-технической информации | 1 |       |                         |             |             |             |             |             |             |             |             |             |             |             |  |