Ориентация монокулярной камеры с использованием точек схождения перспективы

Кудеров П.В.

ИУ9-121

# Оглавление

[Оглавление 2](#_Toc421776670)

[Введение 4](#_Toc421776671)

[1 Аналитическая часть 6](#_Toc421776672)

[1.1 Обзор систем навигации 6](#_Toc421776673)

[1.1.1 Инерциальные системы 6](#_Toc421776674)

[1.1.2 Спутниковые системы 7](#_Toc421776675)

[1.1.3 Cистемы локального позиционирования 8](#_Toc421776676)

[1.1.4 Системы локального позиционирования с использованием методов компьютерного зрения 9](#_Toc421776677)

[1.2 Методы определения углов ориентации камеры 11](#_Toc421776678)

[1.2.1 Метод Хуттунена-Пише 12](#_Toc421776679)

[1.2.2 Другие применения методов поиска точек схождения перспективы 15](#_Toc421776680)

[1.2.3 Краткий обзор аналогичных методов на основе исследования точек схождения перспективы 15](#_Toc421776681)

[2 Научно-исследовательская часть 20](#_Toc421776682)

[2.1 Геометрия перспективных изображений 20](#_Toc421776683)

[2.1.1 Однородные координаты 21](#_Toc421776684)

[2.1.2 Связь между мировой системой координат и системой координат камеры 23](#_Toc421776685)

[2.1.3 Модель камеры–обскуры 26](#_Toc421776686)

[2.2 Выделение сегментов линий на изображении методом Джиои 30](#_Toc421776687)

[2.3 Кластеризация сегментов линий методом Seq–RANSAC 36](#_Toc421776688)

[2.4 Уточнение ТСП 40](#_Toc421776689)

[2.4.1 Уточнение матрицы поворота камеры относительно мировой системы координат 42](#_Toc421776690)

[2.4.2 Определение ориентации по ТСП 44](#_Toc421776691)

[3 Проектно-конструкторская часть 47](#_Toc421776692)

[3.1.1 Выделение сегментов линий 47](#_Toc421776693)

[3.1.2 Конвертация типов данных и предподсчет перед этапом кластеризации 50](#_Toc421776694)

[3.1.3 Вычисление точек схождения перспективы 51](#_Toc421776695)

[3.1.4 Уточнение матрицы поворота на основе полученных направлений ТСП 53](#_Toc421776696)

[4 Технологическая часть 54](#_Toc421776697)

[4.1 Методология разработки и используемые средства 54](#_Toc421776698)

[4.1.1 Библиотека алгоритмов компьютерного зрения OpenCV 54](#_Toc421776699)

[4.2 Руководство пользователя 55](#_Toc421776700)

[4.2.1 Приложение Gyrocam 55](#_Toc421776701)

[4.2.2 Обычный режим работы 56](#_Toc421776702)

[4.2.3 Режим тестирования набора изображений 62](#_Toc421776703)

[4.3 Тестирование приложения 62](#_Toc421776704)

[4.3.1 База данных изображений YorkUrbanDB 62](#_Toc421776705)

[4.3.2 Конфигурация тестирования 63](#_Toc421776706)

[4.4 Результаты тестирования 63](#_Toc421776707)

[4.4.1 Результаты тестирования метода на изображениях «внутри помещения» 63](#_Toc421776708)

[4.4.2 Результаты тестирования метода на изображениях «вне помещения» 64](#_Toc421776709)

[5 Организационно-экономическая часть 66](#_Toc421776710)

[5.1 Введение 66](#_Toc421776711)

[5.2 Организация и планирование процесса разработки программы 67](#_Toc421776712)

[5.2.1 Техническое задание 67](#_Toc421776713)

[5.2.2 Расчёт стоимости проекта 67](#_Toc421776714)

[5.2.3 Затраты на выплату исполнителям 70](#_Toc421776715)

[5.2.4 Определение количества исполнителей 79](#_Toc421776716)

[5.2.5 Календарный график выполнения работ 80](#_Toc421776717)

[5.3 Расчёт сметы затрат 82](#_Toc421776718)

[5.3.1 Затраты на выплату исполнителям 83](#_Toc421776719)

[5.3.2 Суммарные затраты 86](#_Toc421776720)

[5.4 Вывод 87](#_Toc421776721)

[Заключение 88](#_Toc421776722)

[Библиография 89](#_Toc421776723)

[Дополнительные иллюстрации 93](#_Toc421776724)

[Приложение 1. Диаграмма Ганта выполняемых работ 97](#_Toc421776725)

# Введение

Последние десятилетия наблюдается высокий уровень интереса к автономной самоуправляемой технике, и с каждым годом он продолжает расти. Робототехническим системам можно найти применение в условиях военных конфликтов, экстремальных для человека или несопоставимых с его габаритами, в качестве транспортных средств для пассажиро- и грузоперевозок, для проведения масштабных и/или простых шаблонных работ и так далее. В числе наиболее проблемных подсистем самоуправляемой техники значится система навигации, и, в частности, подсистема позиционирования.

Одна из подзадач, возникающих при построении системы позиционирования с использованием алгоритмов компьютерного зрения — определение углов наклона и поворота камеры, то есть создание аналога гироскопа.

В данной работе рассмотрен и реализован метод, предложенный Вилле Хуттуненом и Робертом Пише [**Link1**], который заключается в определении трехмерной ориентации монокулярной камеры с использованием точек схождения перспективы (ТСП), обнаруженных на изображениях.

Создатели метода выделяют следующий набор его достоинств **[Link1]**:

* + изобилие 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
  + робастность относительно случайных нестационарных объектов, попадающих в кадр (люди, транспортные средства и т.п.)

К недостаткам можно отнести тот факт, что тестирование метода проводилось в камеральных условиях с наличием гироскопа.

Целью настоящей работы является реализация и перенос метода Хуттунена — Пише в условия городской застройки. Для тестирования результатов выбран набор тестовых данных YorkUrbanDb [**Link2**], созданный на базе Centre for Vision Research Йоркского университета города Торонто, а также набор случайных изображениях городской среды и сельской местности, взятых из сети интернет или сделанных вручную.

В главе 1 подробно раскрывается проблематика, описаны существующие аналоги, как с использованием методов компьютерного зрения, так и без него. В главе 2 представлена необходимая информация по теоретическим выкладкам, на основе которых реализован метод. Глава 3 содержит описание структуры тестового приложения, реализующего рассматриваемый метод. В главе 4 заострено внимание на конкретных реализациях этапов метода, сделанных решениях, приведены результаты тестирования приложения. В главе 5 описано проведенное исследование по организационно-экономической части, получены результаты стоимости и целесообразности продукта.

Данная работа содержит хх страниц, уу иллюстраций, zz таблиц и qq библиотечных ссылок. Список иллюстраций и таблиц расположен … Список библиотечных ссылок находится …

# Аналитическая часть

## Обзор систем навигации

### 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. Преимущества методов инерциальной навигации состоят в автономности, помехозащищенности и возможности полной автоматизации всех процессов навигации. Основная же проблема — наличие дрейфа, то есть накопление ошибки со временем работы. Различными техниками можно уменьшить величину ошибки, но не избавиться вовсе.

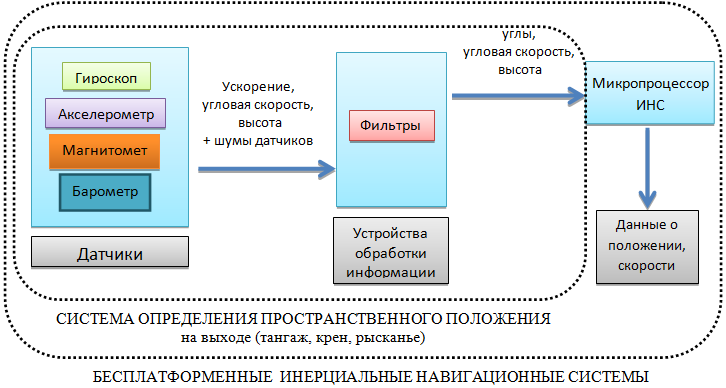


Рисунок  — Схема работы ИНС.

### Спутниковые системы

Не имеют дрейфа глобальные (спутниковые) системы навигации (GPS, GLONASS, GALILEO и другие их аналоги). 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 **[Link34]**.

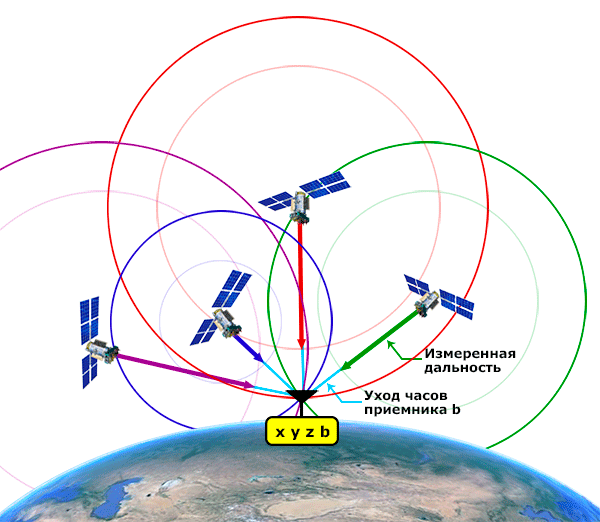


Рисунок  — Схема принципа работы спутниковых систем навигации.

Для существующих систем данного класса характерна погрешность позиционирования порядка 2–15 метров **[Link3]**. Дополнительные проблемы связаны с диапазоном рабочей частоты сигнала — уровень приема сигнала от спутников может серьезно ухудшиться под плотной листвой деревьев, из-за очень большой облачности, в условиях плотной городской застройки. Практически невозможно определить свое точное местонахождение внутри помещений, в тоннеле, причем даже профессиональными геодезическими приемниками **[Link4]**. Но и в благоприятных условиях точности в несколько метров для определенного круга задач может быть недостаточно. В таких случаях глобальные системы навигации используются лишь для локализации нахождения объекта, а уточнение координат объекта ведется с помощью других систем.

### Cистемы 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. Тип используемого сигнала накладывает следующие серьезные ограничения на использование: расстояние до приемников не больше 10 метров, отсутствие препятствий, отражений и помех. Помимо этого, данные системы отличаются высокой стоимостью. Несмотря на то, что в благоприятных условиях достигается очень высокая точность позиционирования порядка 10 сантиметров, такие системы пригодны только в заранее подготовленной (там, где установлены опорные приемники) обстановке.

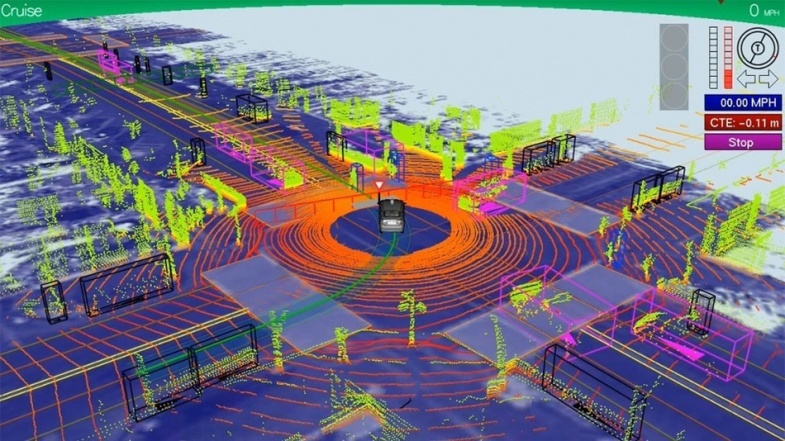


Рисунок  — Пример работы системы позиционирования LIDAR, работающей на автономных автомобилях исследовательского центра компании Google. Стоимость одной системы LIDAR — порядка 70 тыс. долларов .

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.

### Системы локального позиционирования с использованием методов компьютерного зрения

На сегодняшний день видеодатчики имеют маленький размер, энергопотребление и цену, поэтому наблюдается возросший интерес использовать их в системах позиционирования и навигации. Подобные системы на основе анализа изображений не имеют дрейфа, при этом может достигаться довольно высокая точность, сравнимая с ИНС потребительского класса.

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) **[Link5]**, а также метод определения структуры объекта в процессе движения (structure from motion, SfM) **[Link6]**. Оба этих метода в той или иной степени пытаются построить двух- или трехмерную модель окружающей среды, относительно которой происходит движение камеры.



Рисунок  — Пример построенной карты местности в процессе движения робота с использованием SLAM–метода

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 **[Link7]**.

В дополнение к озвученным последним этапом популярно использование метода корректировки пучков (bundle adjustment). Он позволяет повысить качество реконструкции карты окружающей среды и оценки позиционирования камеры, давая совместно оптимальную оценку. Метод принимает в расчет полученные оценки положения и позиции ориентиров, а также параметры относительного движения и оптические характеристики используемой камеры **[Link8]**.

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

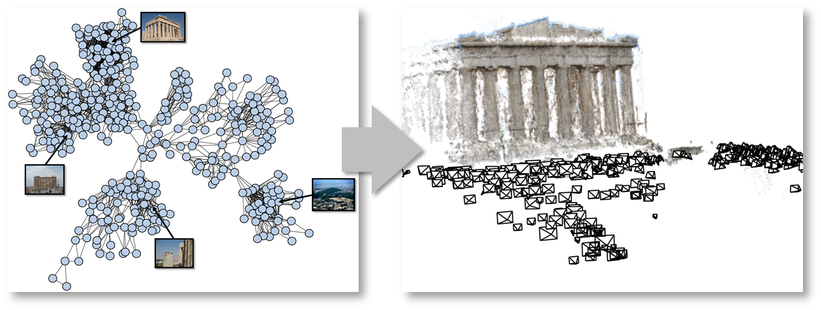


Рисунок  — Пример воссозданных трехмерных наблюдаемых объектов с использованием SfM–метода

## Методы определения углов ориентации камеры

На текущий момент распространены методы, решающие единую задачу навигации и позиционирования целиком. Такие методы получаются очень громоздкими и страдают от невысокой точности и скорости. Однако можно пойти другим путем — разбить задачу на несколько независимых подзадач и решать их отдельно, подобрав под каждую из них наиболее подходящий метод.

Одной из таких подзадач, возникающих при построении системы позиционирования, является задача определения углов наклона и поворота камеры, то есть создание аналога гироскопа.

### Метод Хуттунена–Пише

Метод Хуттунена–Пише заключается в определения трехмерной ориентации монокулярной 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 **[Link1]**:

* 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 Джиои и др. **[Link9]**.

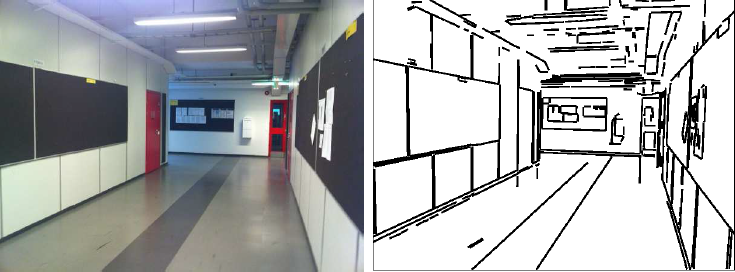


Рисунок  — Пример результата работы детектора сегментов линий Джиои. Слева: оригинальное изображение; справа: обнаруженные сегменты линий.

Среди полученного набора СЛ выделяются три наибольших кластера, задающих линии, сходящихся в трех точках, называемых ТСП. Данный этап проводится с использованием адаптивного алгоритма RANSAC **[Link10]**.

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

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 . 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 **[Link11]**.

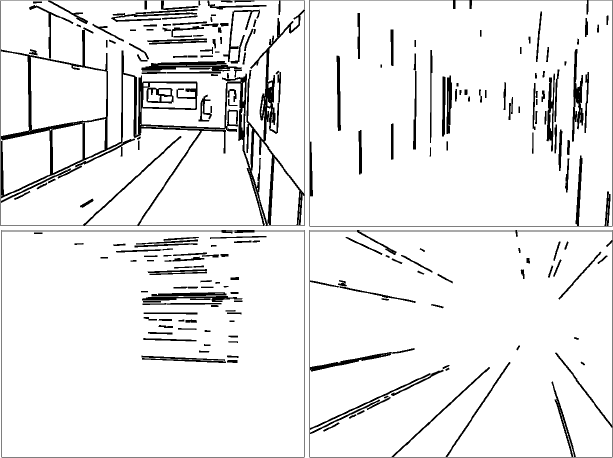


Рисунок  — Пример результата кластеризации алгоритмом RANSAC. Слева сверху: все обнаруженные на изображении СЛ; остальные: кластеры трех доминирующих ТСП, которым соответствуют три взаимно перпендикулярных направления.

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 довычисляются 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 **[Link11]**.

Полученная ортонормированая система векторов задает матрицу поворота камеры, по которой можно вычислить ее углы ориентации.



Рисунок  — Пример результата работы созданной в рамках работы реализации метода на изображении городской местности. Слева: оригинальное изображение; справа: результат, где цветами промаркированы кластеры СЛ, относящиеся взаимно ортогональным ТСП. Полученные углы ориентации: крен —  градуса, тангаж —  градусов, рыскание —  градус.

### 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 **[Link12, Link13, Link14, Link15]**, для корректировки фотографий **[Link16]** или для определения внутренних параметров камеры, то есть для ее калибровки **[Link17, Link18, Link19, Link20, Link21, Link22]**.

### Краткий обзор аналогичных методов на основе исследования точек схождения перспективы

Существует довольно большое число аналогичных работ, посвященных поиску точек схождения перспективы, для решения различных задач. Данные работы можно условно разделить на три категории **[Link30]**. Первые две требуют знания внутренних параметров камеры, а третья работает в условиях некалиброванной камеры.

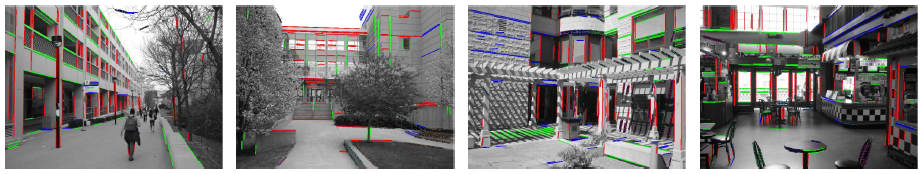


Рисунок  — Пример результатов работы определения ТСП в работе Тардиффа [Link30].

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) **[Link33]**. Довольно новым и только начинающим набирать популярность является метод, предложенный Джиои **[Link9]**. 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 **[Link23]**. Бэрнард и др. **[Link24]** предложили использование преобразования Хафа на квантованной сфере Гаусса, однако позднее было обнаружено, что это часто приводит к нахождению ложных ТСП **[Link25]**. С тех пор большинство работ основываются на параллельности или ортогональности преобладающих на изображении структур с целью исключить ложные срабатывания.

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 **[Link26, Link27]**. Алгоритмы данного типа решают задачу оценки векторов так называемых Манхэттенских направлений (направлений трех взаимно перпендикулярных ТСП), что эквивалентно определению ориентации камеры. Разработаны методы даного типа, основанные на использовании или градиентов сегментов линий **[Link28]**, или самих СЛ **[Link29]**, 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 **[Link31]**. 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 **[Link25, Link35]**.

Другой способ предполагает использование алгоритма максимизации ожидания (Expectation Maximization, EM) для структур, представленных на Гауссовой сфере **[Link12]**, который расширяем на случай некалиброванной камеры **[Link21]**. Данный подход с использованием EM–оптимизации требует предвычисленных грубых оценок ТСП. 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.

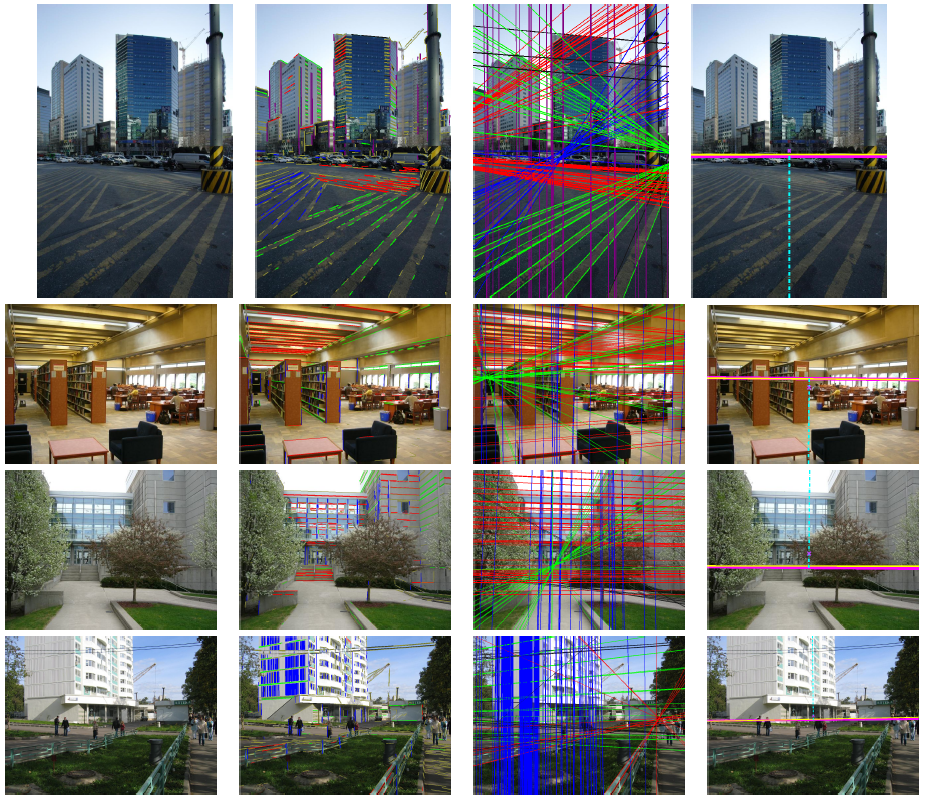


Рисунок  — Пример результата работы метода, предложенного Третьяковой и др. [Link35]. Для каждого входного изображения (крайние слева) слева направо приведены полученные: сегменты линий, кластеры линии, сгруппированные по направлениям, результаты оценки горизонта и зенита.

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 **[Link32]**. 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 количесве 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 **[Link33]**. Данный метод призван решить проблему обоих методов будучи быстрым и оптимальным. 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. К минусам метода можно отнести то, что он сложнее в реализации, чем RANSAC.

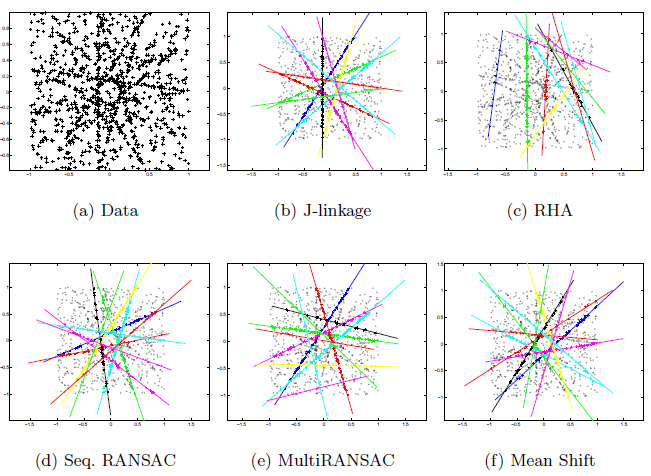


Рисунок  — Сравнение методов кластеризации по результатам работы на изображении одиннадцатиконечной звезды с параметрами гауссова зашумления ( и числом точек шума, равным половине от общего числа [Link33].

# Научно-исследовательская часть

В подразделе 1.2.1 уже приведено краткое описание метода Хуттунена–Пише, которое в грубом приближении состоит из следующих этапов:

* 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

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.

## 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.

### Однородные координаты

В проективной геометрии точки представлены в однородных координатах. Например, рассмотрим точку в двухмерном евклидовом пространстве . Чтобы представить ее на проективной плоскости, необходимо лишь добавить третью компоненту , равную , т.е. .

Для любого ненулевого считается верным равенство . В случае нулевого координаты точки вырождаются в точку , которая не включена в .

Рассмотрим уравнение прямой на двухмерной евклидовой плоскости:

|  |  |  |
| --- | --- | --- |
|  |  | () |

Точки , и только они, удовлетворяющие данному уравнению, являются лежащими на данной прямой. Теперь заметим, что уравнение (39) можно переписать следующим образом:

|  |  |  |
| --- | --- | --- |
|  |  | () |

Таким образом мы определили все точки проективного пространства , лежащие на прямой. Заметим также, что , где , , a 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:

|  |  |  |
| --- | --- | --- |
|  |  | () |
|  |  | () |
|  |  | () |

где оператор  — векторное, а  — скалярное произведение.

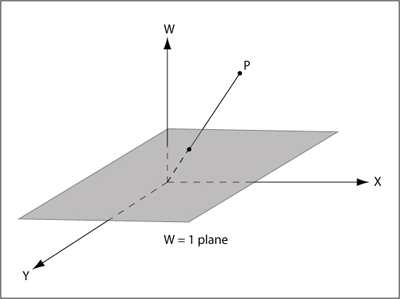


Рисунок  — Связь между точками и можно изобразить как центральную проекцию точки на плоскость .

Чтобы вернуться из обратно в , достаточно поделить координаты точки на z–координату, то есть:

Из данной процедуры сразу видно, что , так как содержит элементы с . Эти точки составляют довольно важное подмножество и называются идальными. 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 можно поставить в соответствие прямую, проходящую через начало координат и точку с выколотой точкой . Подобным образом линия на проективной плоскости может быть визуализирована в плоскостью, образованной началом координат и прямой, заданной перпендиляром . Тогда точкам с координатами в соответствует плоскость , идеальным точкам соответствуют точки на , а идеальной прямой — сама плоскость .

Данная связь между и может быть легко продолжена до связи между и добавлением к точкам четвертой координаты . Такая связь очень хорошо подходит для описания преобразования проецирования трехмерной сцены на двухмерную плоскость изображения.

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

Рассмотрим точку . Для того, чтобы осуществить ее перенос на вектор , мы можем воспользоваться следующим матричным выражением:

|  |  |  |
| --- | --- | --- |
|  |  | () |

Похожим образом выражается поворот точки , заданный матрицей поворота (7), и умножение каждой из координат точки на независимый коэффициент (8):

|  |  |  |
| --- | --- | --- |
|  |  | () |

|  |  |  |
| --- | --- | --- |
|  |  | () |

Можно заметить, что в данных выражениях точка представлена в нормализованных координатах . Также очевидно, что данные выражения остаются верными и для точкек вида :

Интересно рассмотреть, как данные преобразования влияют на идеальные точки . Простой подстановкой проверяется, что:

* Перенос на вектор оставляет идеальную точку на месте
* Поворот действует на идеальную точку абсолютно также, как и на конечную
* Масштабирование на вектор 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 , а матрица поворота связывает соответствующие оси систем, тогда выражения связи имеет вид:

|  |  |  |
| --- | --- | --- |
|  |  | () |

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

Расскроем скобки справа:

и воспользуемся нормализованными координатами:

|  |  |  |
| --- | --- | --- |
|  |  | () |

Тогда матрица , определенная как:

|  |  |  |
| --- | --- | --- |
|  |  | () |

задает матрицу перехода между системами координат.

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

### Модель камеры–обскуры

Одна из самых простых и обычно используемых моделей для конечной проективной камеры — модель булавочного отверстия или модель камеры–обскуры (pinhole camera), в которой точки проецируются на двухмерную плоскость по правилу:

или в нормализованных координатах:

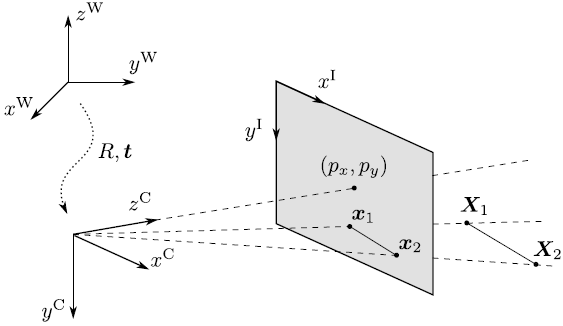


Рисунок  – Модель камеры–обскуры. Поворот камеры и перенос задают преобразование координат из мировой системы в систему, связанную с камерой (9). Имея известную матрицу калибровки камеры , можно задать функию проекции, связывающую координаты точки в мировых координатах в координаты кадра полученного изображения (39).

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

|  |  |  |
| --- | --- | --- |
|  |  | () |

Заметим также, что вектор является нуль–вектором нуль–пространства данного преобразования.

Работая с реальными камерами и точками на полученных с их использованием изображениях, 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 , которая задает внутренние (intrinsic) параметры камеры и предполагается неизменяемой во времени:

|  |  |  |
| --- | --- | --- |
|  |  | () |

где используются следующие обозначения:

* — фокусное расстояние камеры в некоторой единице длины (обычно в мм или дюймах),
* ,  — линейные размеры пикселя, выраженные в той же единице длины, что и . Таким образом и имеют размерность пикселей
* — коэффициент ассиметрии камеры
* ,  — координаты главной точки в пикселях.

Большинство цифровых камер на основе приборов с зарядовой связью (ПЗС) имеют квадратные пиксели (), нулевую ассиметрию () и главную точку, расположенную близко к центру изображения.

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

|  |  |  |
| --- | --- | --- |
|  |  | () |

Рассмотрим прямую в , заданную как где  — некоторая точка на этой прямой,  — направляющий вектор и , . Тогда, воспользовавшись (14), найдем проекцию этой прямой:

|  |  |  |
| --- | --- | --- |
|  |  | () |

так как .

Точка схождения перспективы , соответствующая направлению , является предельной точкой для проекции линии при :

|  |  |  |
| --- | --- | --- |
|  |  | () |

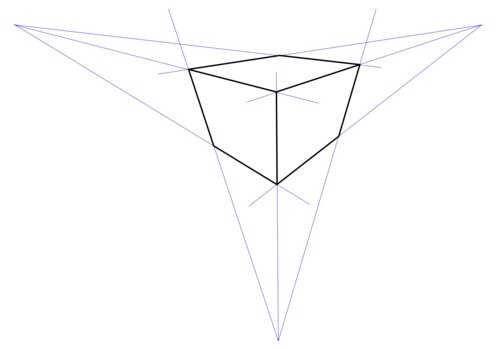


Рисунок  — Пример перспективного изображения куба. В данном случае ТСП, образованные направлениями ребер куба, являются конечными.

В системе координат, связанной с камерой, , поэтому:

|  |  |  |
| --- | --- | --- |
|  |  | () |

Из полученного результата можно заключить следующее:

* ТСП не зависит от положения камеры
* существует взаимно однозначное отношение между и вектором направления прямой в трехмерном пространстве

Будем называть две ТСП ортогональными, если ортогональны векторы направления соответствующих им прямых. ТСП, которые являются идеальными, называются бесконечными, иначе — конечными.

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.

## Выделение сегментов линий на изображении методом Джиои

Первым этапом является выделение сегментов линий на изображении. Для решения было предложено использовать метод Джиои **[Link9]**. Как уже было озвучено в 1.2.3 , 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.

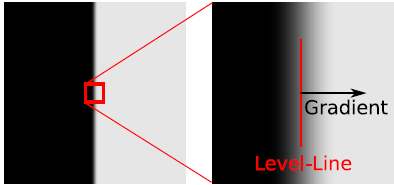


Рисунок  — Градиент изображения и линия уровня

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.

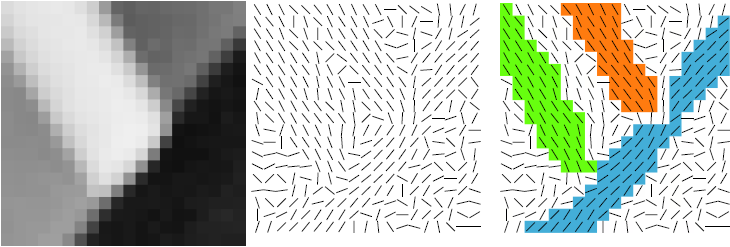


Рисунок  — Зоны поддержки линий. Слева: изображение в градиентах серого; посередине: поле линий уровня; справа: зоны поддержки линий, выделенные цветом.

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 , в которых направление линии уровня совпадает с направлением прямоугольника с учетом некоторого угла толерантности .



Рисунок  — Пример прямоугольника, соответствующего некоторой зоны поддержки линии. Для данного прямоугольника  — количество сонаправленных точек.

Валидация ЗПЛ ведется с использованием принципа Гельмгольца от противного. Для этого сравнением ведется проверка нулевой гипотезы  — отсутствия валидируемой структуры на изображении. Соответственно, зона поддержки линии отвергается, если она имеет недостаточно низкую вероятность в модели гипотезы . Показателем выступает количество сонаправленных пикселей .

Для этого вычисляется вероятность события того, что прямоугольник в модели имеет не меньше сонаправленных точек, чем в наблюдаемой. Пусть  — количество сонаправленных точек для прямоугольника на изображении , а  — общее количество пикселей, в прямоугольнике. Тогда можно рассматривать следующее число событий:

|  |  |  |
| --- | --- | --- |
|  | , | () |

где  — количество тестов или общее количество возможных прямоугольников на рассмотрении,  — случайное изображение, удовлетворяющее модели . Модель нулевой гипотезы фиксирует количество сонаправленных точек , которое зависит от распределения поля линий уровня изображения . То есть является моделью шума для ориентаций градиента изображения, нежели моделью шума самого изображения.

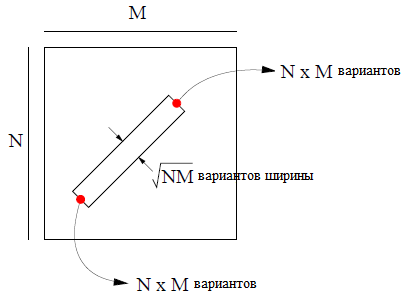


Рисунок  — Оценка количества количества тестов , которая основывается на числе вариантов выбора конечных точек прямоугольника и ширины.

Вероятность равна:

|  |  |  |
| --- | --- | --- |
|  |  | () |

где  — хвост биномиального распределения:

|  |  |  |
| --- | --- | --- |
|  |  | () |



Рисунок  — Пример работы метода LSD на изображении размером пикселей.

Количество прямоугольников на изображении из 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 :

|  |  |  |
| --- | --- | --- |
|  |  | () |

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

|  |  |  |
| --- | --- | --- |
|  |  | () |

Таким образом, когда величина не превышает некоторого порога , прямоугольник некоторой потенциальной СЛ называется –значимым и принимается.

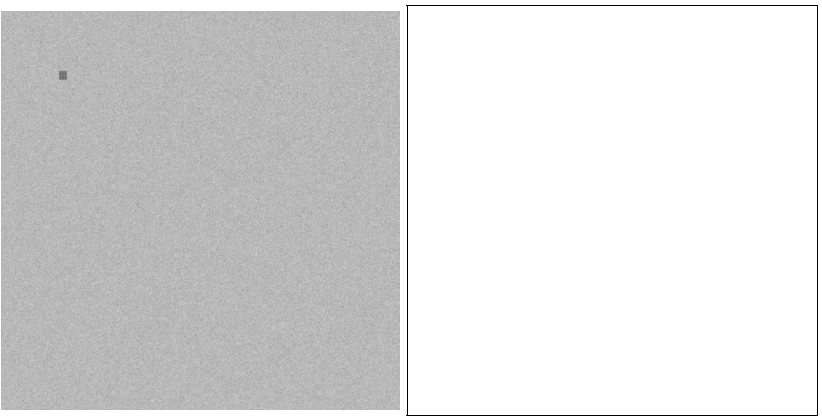


Рисунок  — Пример результата работы метода на маленького квадрата на фоне шума. В данном случае метод работает в соответствии с нашим восприятием — объект слишком мал, чтобы ему придали значение. Размер изображения пикселей.

Среднее число -значимых прямоугольников на изображении шума не превышает , что позволяет контролировать количество ложноположительных срабатываний алгоритма на изображении шума **[Link9]**. Это также является свидетельством того, что данный метод удовлетворяет принципу Гельмгольца. Авторами метода предлагается использовать .

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.

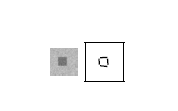


Рисунок  — Пример результата работы метода LSD на участке изображения из Рисунок 18. Теперь, согласно принципу Гельмгольца, квадрат уже является хорошо воспринимаемым объектом и должен быть распознан, что и показано на правой половине рисунка.

## Кластеризация сегментов линий методом Seq. RANSAC

Следующим этапом проводится кластеризация выделенных сегментов линий с использованием последовательного применения адаптивного алгоритма RANSAC (RANdom SAmling Consensus). 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 соответствуюет некоторой ТСП. Всего производится 3 запуска алгоритма.

Алгоритм работает в предположении:

* Параметры могут быть оценены по элементам. В нашем случае параметрами являются СЛ и
* Всего имеется элементов
* Вероятность случайной выборки элемента, являющегося частью искомого кластера равна
* Вероятность завершения алгоритма без нахождения хорошей оценки искомого кластера равна

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 , которая объявляется оценкой потенциальной ТСП
3. Проводится проверка каждого сегмента линии из 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 .
4. Если достаточно велико, полученный кластер принимается и алгоритм завершает свою работу
5. Шаги 1..4 повторяются раз
6. Алгоритм завершается с ошибкой

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

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

|  |  |  |
| --- | --- | --- |
|  |  | () |

где  — точка начала сегмента ,  — норма, учитывающая только первые две координаты, а функции и вычисляются по формулам (3) и (4) соответственно.

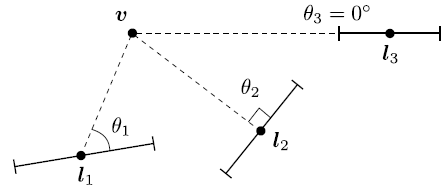


Рисунок  — Расстояние между ТСП и сегментом вычисляется как абсолютная величина угла между сегментом и прямой, соединяющей точку и центр сегмента .

Тогда функция–индикатор принадледжности СЛ кластеру ТСП с допустимой погрешностью задается следующим образом:

|  |  |  |
| --- | --- | --- |
|  |  | () |

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 . Тогда число  — отношение количества внутренних сегментов к числу всех сегментов . Чем выше число , тем более подходящей считается ТСП.

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

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

|  |  |  |
| --- | --- | --- |
|  |  | () |

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

Логарифмируя обе стороны, получаем:

|  |  |  |
| --- | --- | --- |
|  |  | () |

Таким образом на каждой итерации производится переоценка параметра по формуле (39) аппроксимацией снизу значения величиной , 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.

## Уточнение ТСП

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

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

|  |  |  |
| --- | --- | --- |
|  |  | () |

Для всего множества сегментов кластера получается следующая СЛАУ:

|  |  |  |
| --- | --- | --- |
|  |  | () |

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 (39) 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 , которая минимизирует функционал в левой части, а именно:

|  |  |  |
| --- | --- | --- |
|  |  | () |

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 **[Link17]** предлагается преобразовать их в нормированные координаты изображения, соответствующие камере с фокусным расстоянием равным 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 , и заново вычислим уравнения прямых, соответствующих данным сегментам:

|  |  |  |
| --- | --- | --- |
|  |  | () |

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

|  |  |  |
| --- | --- | --- |
|  |  | () |

И задача минимизации (29) превращается в:

|  |  |  |
| --- | --- | --- |
|  |  | () |

Существуют разные методы решения данной задачи оптимизации. В данной работе был выбран метод, использующий сингулярное разложение матрицы . Тогда сингулярный вектор, соответствующий наименьшему ненулевому сингулярному значению матрицы , минимизирует функционал слева из (39).

Полученная точка задает направление ТСП изображения. Далее производится нормировка их координат, т.е. вычисление единичных векторов направлений ТСП.

### 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.

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

Тройка направлений может быть не взаимно ортогональной. В этом случае производится ортогонализация матрицы , составленной из единичных векторов направлений найденных ТСП. Ближайшую в смысле нормы Фробениуса ортогональную матрицу можно вычислить, воспользовавшись сингулярным разложением матрицы . Тогда:

|  |  |  |
| --- | --- | --- |
|  |  | () |

где и определяются из сингулярного разложения матрицы .

Теперь, чтобы составить матрицу поворота 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 следющим образом:

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

В итоге, переставив столбцы матрицы и поправив знаки, мы можем восстановить матрицу поворота :

|  |  |  |
| --- | --- | --- |
|  |  | () |

### Определение ориентации по ТСП

Для определения углов ориентации камеры по имеющейся матрице поворота 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 , и . Пусть углы поворота относительно некоторого набора осей равны , , . Будем называть вектором углов Эйлера следующий вектор:

|  |  |  |
| --- | --- | --- |
|  |  | () |



Рисунок  — Углы Эйлера в нотации (1, 2, 3). Сначала производится поворот относительно оси на угол , затем вокруг оси на угол , и последним осуществляется поворот вокруг оси на угол . Данная последовательность поворотов переводит оси координат в .

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 , и соответственно:

|  |  |  |
| --- | --- | --- |
|  |  | () |

Числа в нотации (1, 2, 3) как раз обозначают индексы осей в произведении матриц поворотов формулы (39). Стоит помнить, что умножение матриц производится справа налево, поэтому первый поворот осуществляется матрицей . Обозначенным углам поворота даны названия: крен, тангаж и рыскание (roll, pitch, yaw). Эти названия берут свое начало из морской навигации.

По имеющейся матрице поворота вектор углов Эйлера вычисляется так **[Link36]**:

|  |  |  |
| --- | --- | --- |
|  |  | () |

где функция  — реализация функции , а двухаргументная функция задается через реализацию функции  —  — следующим образом:

|  |  |  |
| --- | --- | --- |
|  |  | () |

# Проектно-конструкторская часть

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 gyrocam. Для краткости оно опущено в именах структур и глобальных функций. Пространства имен подключаемых библиотек наоборот сохранены для ясности.

## 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 Таблица 1.

Таблица  — 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:  при  при |
| 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 являет 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 Таблица 2. А в приведено описание методов данного класса.

Таблица  — Описание полей типа SingleRunConfig.

| Поле | Тип | Описание |
| --- | --- | --- |
| 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  — Описание методов класса SingleRunConfig.

| Метод | Описание |
| --- | --- |
| 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 Таблица 4, а описание методов — в Таблица 5.

Таблица  — Описание полей класса YorkUrbanDbTestRunConfig.

| Поле | Тип | Описание |
| --- | --- | --- |
| 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  — Описание методов класса YorkUrbanDbTestRunConfig.

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

## 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. initColors();
5. loadImage();
6. loadCalibrationMatrix();
7. extractLineSegments();
8. std::vector<cv::Point3d> vps;
9. for (int i = 0; i < 3; i++)
10. {
11. if (notUsedSegments.size() < 2)
12. break;
13. cv::Point3d refinedVpNormalized = findVanishingPoint(vps);
14. vps.push\_back(refinedVpNormalized);
15. }
16. result.vpBasis = getRotationMatrix(vps);
17. refineVpBasis(result.vpBasis);
18. result.orthoVpBasis = getNearestOrthogonalMatrix(result.vpBasis);
19. reorderColumn(result.orthoVpBasis, 1);
20. reorderColumn(result.orthoVpBasis, 0);
21. reorderColumn(result.orthoVpBasis);
23. result.eulerAngles = getEulerAngles(result.orthoVpBasis.t());
24. result.runTime = timeCounter.GetCounter();
25. saveResults(result);
26. drawProcessedImage(getEulerAnglesString(result.eulerAngles));
27. 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 Таблица 6.

Таблица  — Описание полей класса ImageProcessor.

| Поле | Тип | Описание |
| --- | --- | --- |
| config | SingleRunConfig | Объект специфичных настроек конфигурации. |
| settings | Settings | Объект глобальных настроек конфигурации. |
| calibrationMatrix | cv::Mat | Матрица калибровки . |
| inversedCalibrationMatrix | cv::Mat |  |
| image | cv::Mat | Матрица изображения |
| segments | std::vector<LineSegment> | Список всех распознанных сегменов, прошедших фильтрацию. |
| notUsedSegments | std::set<int> | Множество–индикатор, которое содержит индексы сегментов линий из segments, участвующих в кластеризации. |
| colors | std::vector<cv::Scalar> | Палитра используемых цветов |

Первым этапом процесса обработки является выделение сегментов линий.

### Выделение сегментов линий

Для решения данной задачи в приложении использована реализация детектора сегментов линий Джиои в библиотеке OpenCV версии 3.0.0 beta. 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 2.2 , 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.

*Листинг 2 — Метод 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);

1. segments = LineSegment::toLineSegments(
2. lines\_std,
3. findMinAllowedLineSegmentLength(image)
4. );
5. notUsedSegments.clear();
6. for (int i = 0; i < segments.size(); i++)
7. notUsedSegments.insert(i);

1. if (settings.BUILD\_IMAGE && settings.DRAW\_RAW\_SEGMENTS)
2. drawFoundSegments(segments, image, colors[0], 1);
3. if (settings.TRACE\_ENABLED)
4. 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 зависящего линейно от длины наибольшей из сторон обрабатываемого изображения.

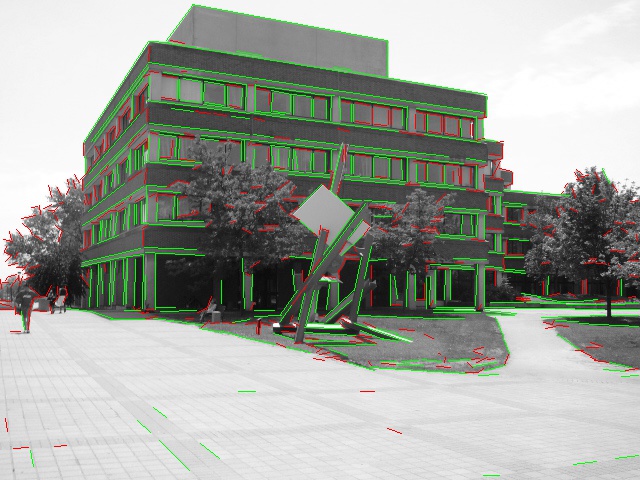


Рисунок  — Пример результата работы детектора сегментов линий LSD библиотеки OpenCV. Зеленым цветом отрисованы сегменты, прошедшие фильтрацию по длине, красным — отброшенные.

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, из-за коротких СЛ страдает результирующая точность. 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 пикселей, поэтому оптимальное значение параметра фильтрации подбиралось именно под него — опытным путем было выбрано значение . Для изображений других размеров величина масштабируется линейно (см. 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 Таблица 6.

Таблица  — Описание полей типа LineSegment.

| Поле | Тип | Описание |
| --- | --- | --- |
| origin | cv::Vec4i | Оригинальная четверка координат концов в нотации . |
| 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.

### Вычисление точек схождения перспективы

В описании метода в подразделе 1.2.1 и разделах 2.2 – 2.3 этапы уточнения и определения направления ТСП описаны после этапа кластеризации. Однако данные этапы независимы для разных ТСП, и поэтому 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 gyrocam, тестового приложения. Список методов данного пространства имен приведен в Таблица 7 — Описание методов пространства имен ransac..

Таблица  — Описание методов пространства имен ransac.

| Метод | Описание |
| --- | --- |
| int countInducedSegments(  const std::vector<LineSegment> &segments,  const std::set<int> &notUsed,  const cv::Point3d &vp,  double angleEpsilon  ) | 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). |
| std::vector<int> getInducedSegments(  const std::vector<LineSegment> &segments,  const std::set<int> &notUsed,  const cv::Point3d &vp,  double angleEpsilon  ) | 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). |
| std::vector<LineSegment> resolveIndices(  const std::vector<LineSegment> &segments,  const std::vector<int> &indices  ) | Вспомогательный метод, разыменовывающий список индексов сегментов в список самих объектов сегментов линий типа LineSegment. |
| void markInducedSegmentsAsUsed(  std::set<int> &notUsed,  const std::vector<int> &toErase  ) | Вспомогательный метод, удаляющий из множества–индикатора notUsed список индексов СЛ toErase. |
| std::vector<LineSegment> nextCluster(  const std::vector<LineSegment> &segments,  const std::set<int> &notUsed,  cv::Point3d &outVanishingPoint  ) | 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 функции–индикатора принадлежности сегмента кластеру (24) опытным путем выбрано значение .

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. cv::Mat res = cv::Mat::zeros(3, 1, CV\_64FC1);
7. solveZ(A, res);
8. res = res.t();
9. return cv::Point3d(res);
10. }

В данной процедуре для решения задачи минимизации (32) используется существующая реализация метода сингулярного разложения матрицы библиотеки OpenCV, которой отвечает класс 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 сингулярного разложения, то есть — сингулярные числа. В таком случае решением задачи минимизации матрицы уравнения (32) будет вектор–строка матрицы vt, соответствующий последнем 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 переводится обратно в ненормированные координаты изображения:

|  |  |  |
| --- | --- | --- |
|  |  | () |

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.

Всего производится две итерации уточнения, которые описаны в листинге 5 метода ImageProcessor::smartVpRefinement. Заметим, что эта процедура производится с меньшим углом толерантности , 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.



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

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:smartVpRefinenement*

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);
19. --steps;
20. }
21. while (steps > 0 && cluster.size() > 3);
23. ransac::markInducedSegmentsAsUsed(notUsedSegments, indices);
24. drawInducedCluster(vp, cluster, colors[i], settings.BUILD\_IMAGE);
25. return normalizedVp;
26. }

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

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

### Уточнение матрицы поворота на основе полученных направлений ТСП

Полученные направления ТСП становятся векторами–стоблцами матрицы , 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 , столбцы полученной матрицы необходимо переупорядочить и проверить их ориентацию в соответствии с процедурой, описанной в 2.4.1 . 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 (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 , полученную из векторов-столбцов направлений ТСП из предыдущих этапов вычислений. Таблица 10 содержит матрицу , полученную в результате выполнения текущего этапа. Соответствие направлений ТСП из Таблица 9 и осей координат показано добавленными верхними индексами.

Таблица  — Пример матрицы .

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 0,006024761727378262 | 0,9192487276424464 | 0,3948734198365318 |
| -0,999897274421663 | 0,02666194976707568 | -0,03134785016268479 |
| -0,01300549308015694 | -0,392773366157077 | 0,9182005742738241 |

Таблица  — Пример матрицы .

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 0,9190373574139783 | -0,00909484524039407 | 0,394065501493831 |
| 0,01867073538966751 | 0,9996160538067567 | -0,02047311924978573 |
| -0,3937280016934789 | 0,02617305411793219 | 0,9188543039680444 |

### 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. a = a \* 180 / CV\_PI;
8. return a;
9. }

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 .

Таблица  — Описание полей структуры SingleRunResult.

| Поле | Тип | Описание |
| --- | --- | --- |
| vpBasis | cv::Mat | Матрица единичных векторов–столбцов направлений найденных 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. |

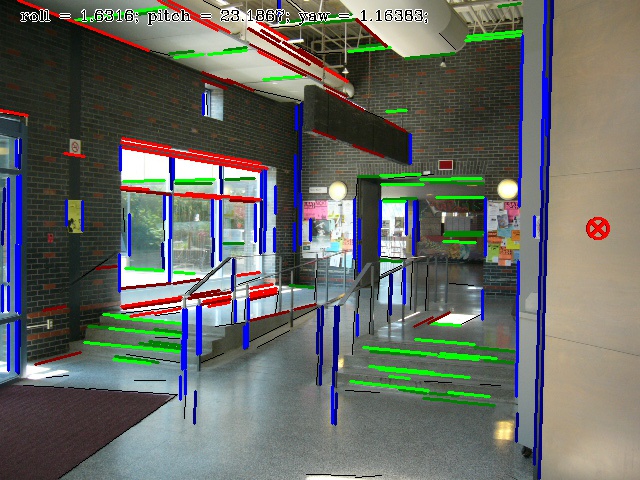


Рисунок  — Результат работы этапа вычисления углов Эйлера в нотации (1, 2, 3) на примере кадра, снятого внутри помещения. Сверху на изображении строка со значениями полученных углов: крен , тангаж , рыскание .



Рисунок  — Пример результата работы приложения на снимке вне помещения. Слева: оригинальное изображение; справа: изображение, полученное в результате вычисления углов ориентации.

## 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).

# Технологическая часть

## Методология разработки и используемые средства

Результатом работы является Win32 консольное приложение, реализующее алгоритм 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++. Разработка велась с использованием средств интегрированной среды разработки Visual Studio 2010.

### Библиотека алгоритмов компьютерного зрения OpenCV

При реализации метода активно использовалась библиотека алгоритмов компьютерного зрения с открытым исходным кодом OpenCV версии 3.0.0 beta.

К причинам использования можно отнести, во–первых, широкий набор структур данных для работы с геометрическими и алгебраическими примитивами, такими как точки, матрицы, с поддержкой математических операций над ними. Во-вторых, наличие готовых реализаций части алгоритмов, которые требовалось реализовать при написании приложения. К ним относятся детектор сегментов линий Джиои и метод сингулярного разложения матриц.

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 и компилятора C++, поставляемого со средой разработки Visual Studio 2010.

## Руководство пользователя

### Приложение Gyrocam

Для использования данного приложения требуется установленная на компьютере пользователя библиотека компьютерного зрения OpenCV версии 3.0.0 beta. Описание процесса установки и необходимые файлы можно найти на сайте библиотеки **[Link36]**.

Приложение запускается с набором аргументов. обязательных и опциональных аргументов. Часть из опциональных аргументов являются предварительно заданными метками, начинаются со знака минус ‘-‘ и не зависят от позииции во входном массиве аргументов, остальные – неименованные, позиционно зависимые.

* первый неименованный аргумент, обязателен – строка абсолютного или относительго пути к входному файлу изображения. Поддерживается такие форматы как bmp, jpg, jpeg,tiff, png.
* второй неименованный аргумент, необязателен – строка абсолютного или относительного пути выходного файла изображения. Формат выходного изображения будет выбран на основе расширения в имени файла данного пути. Поддерживаемые форматы те же, что и во входном аргументе. В случае, если аргумент не передан, в качестве выходного пути будет использован путь к входному изображению, а имя файла – путем объединения имени входного изображения без расширения и строки «\_result.jpg» (таким образом выходной файл будет записан в формате jpeg)
* третий неименованный аргумент, необязателен – строка абсолютного или относительного пути к файлу с матрицей калибровки камеры. Формат файла описан далее в данной главе. В случае, если путь не задан, будет использована единичная матрицы калибровки (тоже самое, что не использовать ее вообще)
* именованный аргумент “-noWait”, необязательный – если данный аргумент не передан при запуске, то после завершения работы, приложение не закроет автоматически все окна, а будет ожидать ввода с клавиатуры от пользователя. Такой режим полезен при презентации работы программы в интерактивном режиме. Если же аргумент передан, программа закроет все окна и завершит работу, что удобно при массовом запуске приложения (например, с помощью .bat скрипртов).
* именованный аргумент “-trace”, необязательный – включает трассировку работы приложения, в результате которой в поток вывода приложение будет отправлять промежуточные результаты: координаты точек схождения перспективы (ТСП) до уточнения, матрицы декомпозиции на этапе уточнения ТСП, сравнение уточненной ТСП с ТСП, полученной с использованием встроенного метода SVD::solveZ библиотеки OpenCV
* именованный аргумент “-silent”, необязательный – режим тихого выполнения, в котором приложение не вычисляет постобработанное изображение и не отображает окно с ним. Также отменяет трассировку приложения. Полезен при нагрузочном тестировании производительности приложения или массовом запуске с помощью .bat скриптов.
* именованный аргумент “-yorkUrbanDb”, необязательный – декларирует особый режим тестирования базы данных изображений YorkUrbanDb, подробно описанный далее в данной главе. Данный режим автоматически включает тихий режим “-silent”.

Опциональные аргументы не зависят от регистра.

### Обычный режим работы

Для изображения, заданного входным аргументом, приложение рассчитывает единичные векторы ТСП и выводит окно. В данном окне отображается само исходное изображение с наложенными поверх него прямыми линиями разных цветов. В случае, если координаты точки схождения перспективы, выраженные в координатах изображения, находятся в пределах окна, данная точка также отрисовывается перечеркнутой окружностью определенным цветом. Пример окна приложения показан на рисунке (Рисунок 25).

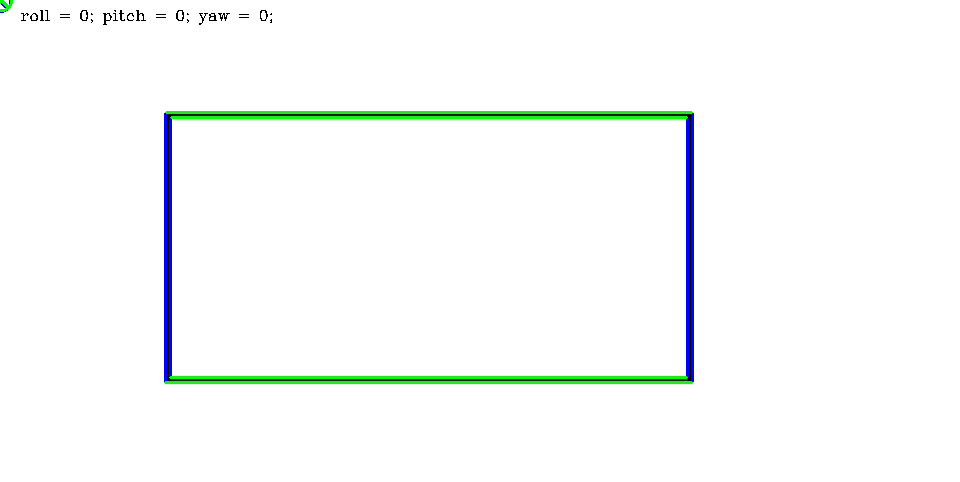


Рисунок  — Пример результата работы приложения на синтетическом тесте, где выделено только два кластера сегментов линий.

Сначала поверх изображения накладываются линии, соответствующие трем наилучшим (с вероятностью >= 95%) кластерам сегментов линий, сходящихся в какой-либо одной точке. Для них используются темные варианты трех цветов – темно-синий, темно-зеленый и темно-красный цвета в порядке выделения каждого кластера.

Далее поверх полученного изображения накладываются линии, соответствующие кластерам сегментов линий, сходящихся в вычисленных ТСП. Данные линии отображаются яркими вариантами трех цветов – синим, зеленым и красным (для сравнения, например, если темный вариант красного – bgr(0, 0, 127), то яркий – bgr(0, 0, 255)). Кластеры одного цвета, но разной степени яркости в общем случае могут как полностью совпадать, так и частично различаться. Связь между ними следующая – сегменты темного цвета как бы продуцируют некоторую ТСП, а яркого – индуцированы этой же ТСП. В случае полного совпадения кластеров одного цвета – поверх кластера темного цвета будет наложен кластер яркого цвета и он не будет виден.

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

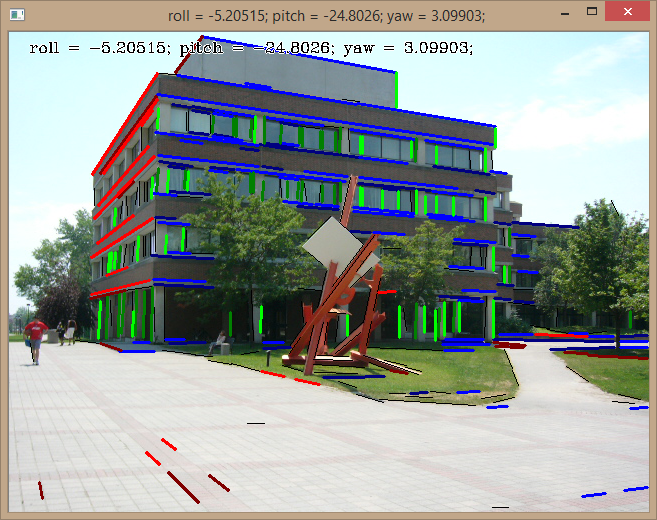


Рисунок – Окно с обработанным изображением, поверх которого наложены сегменты линий, соответствующие найденным ТСП. Сверху указаны полученные углы ориентации камеры в градусах.

Помимо описанного выше окна с изображением после вычисления ТСП приложение выводит в консоль некоторую служебную информацию от модуля opencv и затем две матрицы 3х3 с числами с плавающей запятой (Рисунок 26). Каждая матрица имеет формат как в примере ниже:

[0.99999738, 0.0022501017, 0.00047868371;

0.93076748, 0.36539984, -0.012442469;

-0.10985497, -0.99394757, -0.00042265668]

Границы матрицы задаются квадратными скобками, строки матрицы разделяются точкой с запятой с последующим переводом строки, числа в рамках одной строки разделяются запятыми.

Первой матрице соответствуют единичные векторы найденных ТСП. Каждый вектор представляет собой строку матрицы. Порядок соответствия векторов ТСП наложенным на изображение линиям следующий – синий, зеленый, красный.

Вторая матрица – ближайшая к первой ортонормированная матрица в смысле нормы Фробениуса. Данная матрица является искомой матрицей поворота системы координат камеры.

Описанный выше вывод двух матриц также сохраняется на диск текстовый файл в том же формате. Полный путь данного файла получается прибавлением расширения «.txt» к строке пути второго входного аргумента (пути к выходному изображению). Например, если при запуске приложения вторым аргументом на вход передать относительный путь «../TestSamples/output.jpg», то матрицы будут записаны в файл по следующему пути: «../TestSamples/output.jpg.txt».

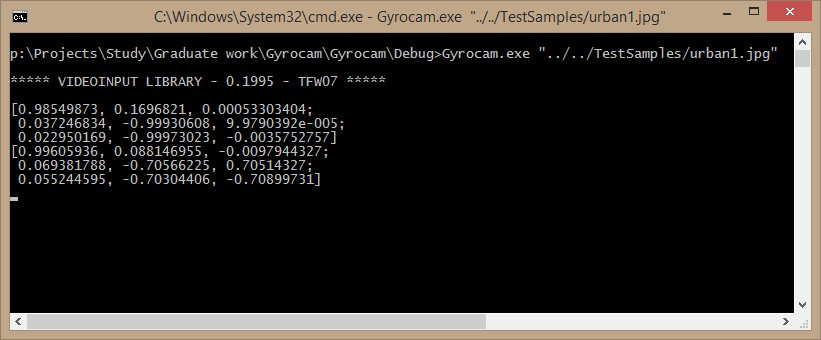


Рисунок – Вывод в консоли матриц направлений ТСП (неортогональной и ортогональной, соответственно) на тестовом запуске.

Пример строки аргументов запуска приложения приведен ниже:

…>Gyrocam.exe ‑noWait ‑trace "../YorkUrbanDB\_indoor/urban.jpg" "../YorkUrbanDB\_indoor/urban\_result.png" "../YorkUrbanDB\_indoor/cameraParameters.txt"

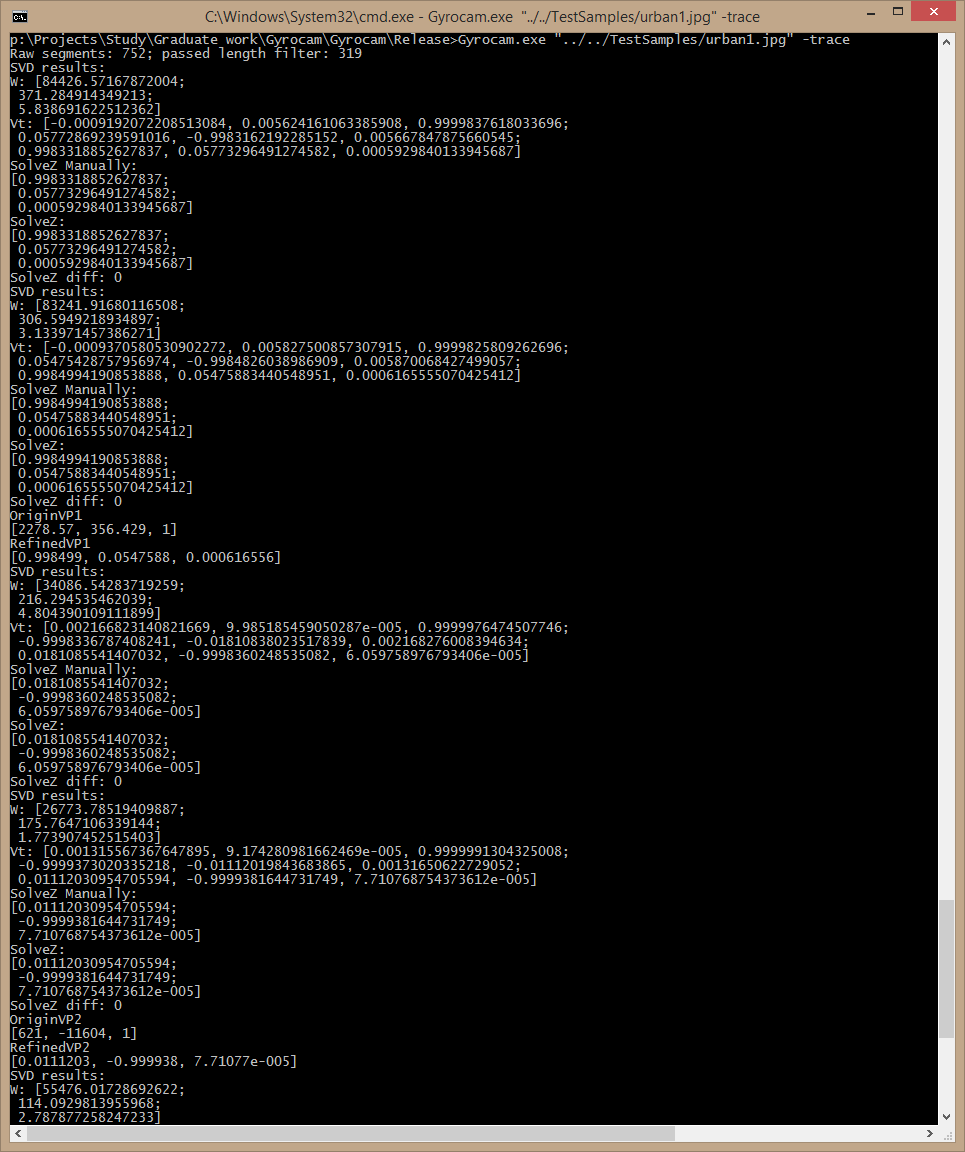


Рисунок  — Пример вывода в консоль в результате работы приложения с включенным режимом трассировки.

### Режим тестирования набора изображений

Режим тестирования набора изображения задатеся флагом ‑yorkUrbanDb.

Имеющаяся архитектура требует в качестве входного аргумента путь к корневой директории набора тестовых изображений. 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 с именем «$.jpg», а также файлы «$GroundTruthVP\_CamParams.mat.txt» и «$GroundTruthVP\_Orthogonal\_CamParams.mat.txt», содержащие матрицы направлений ТСП до ортогонализации и после нее, где вместо доллара подставляется название этой субдиректории.

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.

… Пример строки аргументов:

…>Gyrocam.exe ‑yorkUrbanDb "../YorkUrbanDB\_indoor/" "../YorkUrbanDB\_indoor/indoor.txt" "../YorkUrbanDB\_indoor/cameraParameters.txt"

## Тестирование приложения

### База данных изображений YorkUrbanDB

Описанное консольное приложение было протестировано на наборе изображений YorkUrbanDb (The York Urban Line Segment Database) **[Link2]**. 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

Эти данные доступны в виде файлов данных среды MatLab. Поэтому, для того, чтобы иметь возможность пользоваться ими программно, был написан небольшой скрипт на языке MatLab, осуществляющий конвертацию необходимых данных в текстовый формат.

### Конфигурация тестирования

Тестирование проводилось на личном персональном компьютере Intel Core i7 920, имеющий 4 физических ядра или 8 виртуальных в режиме Hyper–threading. Тактовая частота в нормальном режиме составляет 2,66 Ггц. Оперативная память имеет размер 6 Гб и работает в трехканальном режиме.

## Результаты тестирования

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

В результате тестирования было обработано 45 снимков внутри помещений из коллекции базы данных YorkUrbanDB. Время обработки всех снимков составило сек, что эквивалентно fps.

В качестве меры отклонения от истинных значений было принято решение использовать вектор, составленный из углов между соответствующими истинными и полученными направлениями ТСП. Углы измерялись в градусах.

Вектор среднего отклонения углов полученных направлений ТСП от истинных составил:

Вектор среднеквадратичного отклонения составил:

По каждой из осей:

Результаты распределения полученных направлений представлены на диаграммах с разделе дополнительных иллюстраций (Диаграмма 1 и Диаграмма 2).

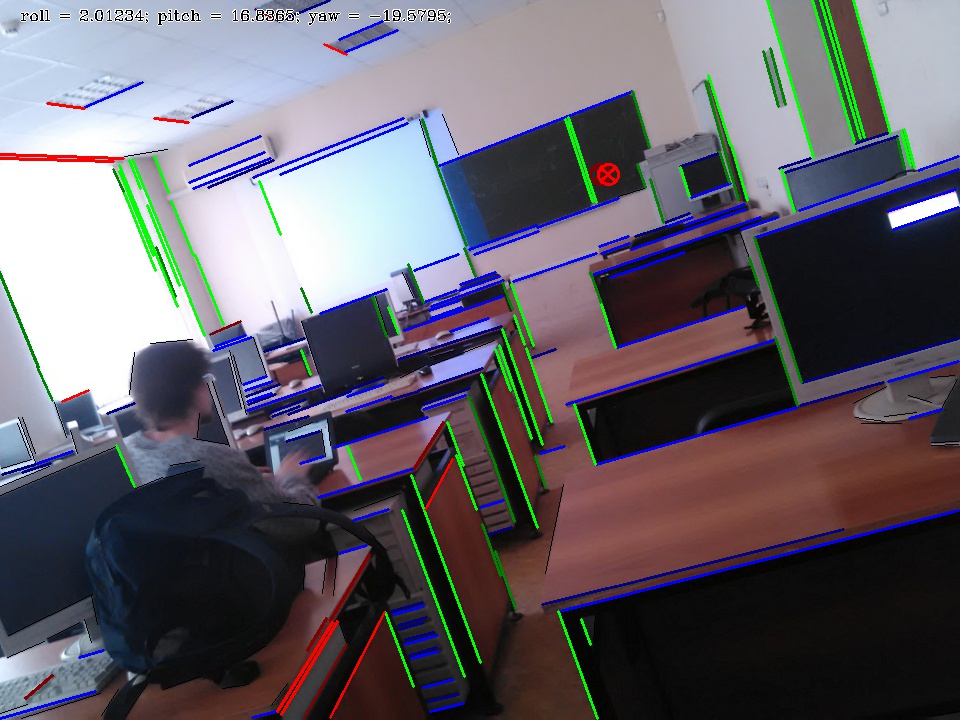
****

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

### 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. Время обработки всех снимков составило сек, что эквивалентно fps.

Вектор среднего отклонения углов полученных направлений ТСП от истинных составил:

Вектор среднеквадратичного отклонения составил:

По каждой из осей:

Результаты распределения полученных направлений представлены на диаграммах с разделе дополнительных иллюстраций (Диаграмма 3 и Диаграмма 4).

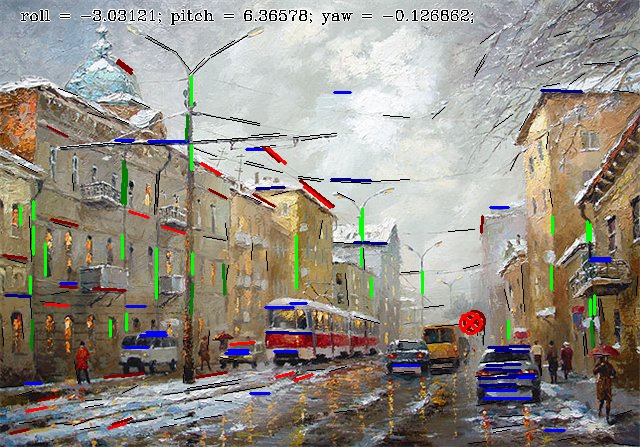


Рисунок  — Пример результата работы приложения на синтетическом тесте изображения картины, на которой художник сделал попытку соблюсти правила перспективы. Как видно, кластеризация прошла достаточно успешно.

# Организационно-экономическая часть

## Введение

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.

## Организация и планирование процесса разработки программы

### Техническое задание

Для расчёта затрат на выполнение дипломного проекта используем техническое задание, представленное в таблице 2.

Таблица 2 – Техническое задание

| № | Наименование | Значение |
| --- | --- | --- |
| 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 |

### Расчёт стоимости проекта

Базовой экономической характеристикой для оценки дипломного проекта является её стоимость. Стоимость дипломного проекта, как экономическая категория, определяется по формуле 1 [ссылка].

| , | (1) |
| --- | --- |

где

* + – стоимость проектных работ,
  + – себестоимость проектных работ,
  + – прибыль.

Выразим прибыль через себестоимость работ в формуле 2:

| , | (2) |
| --- | --- |

где – желаемый для исполнителя уровень рентабельности.

Исходя из формул 1 и 2, можем выразить стоимость проекта как показано в формуле 3:

| , | (3) |
| --- | --- |

Таким образом для расчёта стоимости проекта потребуется спрогнозировать себестоимость. Для её определения необходимо рассчитать каждую стадию сметы затрат по формуле 4 [ссылка]:

| , | (4) |
| --- | --- |

где соответствующие представлены в таблице 3.

Таблица 3 – Структура сметы затрат на выполнение проекта

| № | Наименование статьи затрат | Затраты, % |
| --- | --- | --- |
| 1 | Материальные | 20 |
| 2 | Заработная плата (основная и дополнительная) | 45 |
| 3 | Отчисления на социальные нужды |
| 4 | Амортизация оборудования | 20 |
| 5 | Прочие затраты | 15 |

Прочие затраты обычно составляют 15% [ссылка] 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 и через это значение определить себестоимость проекта. Исходя из значений доли статьи расходов к общей стоимости проекта и из формулы 4, рассчитаем себестоимость проекта в формуле 5:

|  | (5) |
| --- | --- |

### Затраты на выплату исполнителям

В заработную плату включается основная и дополнительная заработная плата всех исполнителей, непосредственно занятых разработкой, с учетом их должностного оклада и времени участия в разработке. Дополнительную заработную плату в составе обобщённой определим в формуле 6 [ссылка]:

| , | (6) |
| --- | --- |

где

* + – заработная плата,
  + – основная заработная плата,
  + – дополнительная заработная плата,
  + – коэффициент отчислений на дополнительную заработную плату.

Коэффициент примем равным 100%, учитывающим расходы на очередные отпуска, выплаты за выслугу лет и прочее. Таким образом, подставив это значение в формулу 6, получим значение заработной платы, определённой в формуле :

| , | (7) |
| --- | --- |

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, [ссылка] указаны в таблице 4.

Таблица 4 – Ставки страховых взносов на 2015 год

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

Рассчитаем отчисления на социальные нужды с заработной платы в формуле (8):

|  | (8) |
| --- | --- |

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 по формуле 9:

| , | (9) |
| --- | --- |

где

* + – среднедневный ФОТ рабочих i-й специальности,
  + – количество дней работ для рабочих i-й специальности,
  + – количество рабочих i-й специальности.

Минимальный размер оплаты труда (МРОТ) устанавливается соответствующим законом [ссылка] и составляет 5965 рублей. С учетом принятого в организации тарифного коэффициента на работы (KT ≈ 10.9) размер оплаты труда инженера по защите информации и специалиста по защите информации первой категории составляет:

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 5 [ссылка]:

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

Таблица 5 – Этапы разработки программного продукта

| № этапа | Название этапа | Общий состав работ этапа |
| --- | --- | --- |
| 1 | Техническое задание (ТЗ) | Разработка ТЗ. |
| 2 | Эскизный проект (ЭП) | Исследование существующего программного продукта.  Уточнение структуры и формы представления входных и выходных данных. Разработка алгоритма решения задачи. Разработка структуры программы.  Разработка пояснительной записки. 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.  Разработка, создание и утверждение порядка и методики испытаний, корректировка программы. |
| 5 | Внедрение (В) | Разработка программной документации.  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 [ссылка] по формуле 11. За единицу нормирования принимается разработка одного листа технической документации формата A4 эскизного проекта, что составляетот общей трудоёмкости проекта, согласно [ссылка].

| , | (11) |
| --- | --- |

где

* + – трудоёмкость всего проекта, чел/час;
  + – количество требуемых листов документации, шт;
  + – норма времени на разработку одного листа формата А4, час/шт;
  + – коэффициент, учитывающий объём входной информации;
  + – коэффициент, учитывающий сложность контроля информации;
  + – коэффициент, учитывающий вид обработки информации (режим обработки информации);
  + – поправочный коэффициент по степени применения типовых проектных решений, пакетов прикладных программ, типовых проектов, типовых программ и стандартных модулей;
  + – коэффициент учёта уровня алгоритмического языка программирования;
  + – доля трудозатрат в общем проекте.

Коэффициент, учитывающий объём входной информации зависит от количества наборов входных данных и выражается по формуле 12:

| , | (12) |
| --- | --- |

где

* , и – значения коэффициентов учета вида используемой информации для переменной, нормативно-справочной информации и баз данных соответственно;
* , и – 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 6. А по степени сложности алгоритма к одной из трёх групп, указанных в таблице 7.

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 [ссылка]: ; ; . Подставив их в уравнение 12 получим значение в уравнении 13:

| , | (13) |
| --- | --- |

Таблица 6 – Классификация степени новизны разрабатываемого программного продукта

| Название группы | Описание |
| --- | --- |
| А | 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 7 – Классификация степени сложности алгоритма программной продукции

| Степень сложности | Описание |
| --- | --- |
| 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 уравнении 11 коэффициенты. Количество требуемых листов документации исходя из технического задания равно . Норма времени на разработку документации одного листа формата А4 равно часам [ссылка]. Согласно таблицам [ссылка] коэффициент учёта режима обработки информациидля технического проекта с обработкой информации в реальном времени группы новизны «В» равен . Коэффициент, учитывающий сложность контроля информации, для данной специфики задачи равен . 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 . Программный код разрабатывается на языке высокого уровня, поэтому коэффициент учёта уровня алгоритмического языка программирования равен .

Таким образом, подставив коэффициенты в уравнение 11 получим трудоёмкость рабочего проекта в формуле 14:

| , | (14) |
| --- | --- |

Для того, чтобы определить количество человек, требуемых для выполнения каждого из этапов разработки, необходимо определить трудоёмкость каждого этапа. В таблице 8 представлены доли трудоёмкости каждого этапа, согласно [ссылка], и рассчитанное по формуле 15 абсолютное его значение.

| , | (15) |
| --- | --- |

где – доля каждого этапа.

Таблица 8 – Трудоёмкость этапов разработки программного продукта

| № 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 определяется по формуле 16:

| , | (16) |
| --- | --- |

где

* – трудоёмкость этапа, чел/час;
* – коэффициент дополнительных работ, учитывающий затраты времени на работы, не предусмотренные нормативами, ;
* – фонд рабочего времени исполнителя за период, определяемый сроками;
* – коэффициент, учитывающий выполнение норм, .

Фонд рабочего времени каждого исполнителя за период с 10 марта 2015 года по 31 мая 2015 года рассчитывается по формуле 17:

| , | (17) |
| --- | --- |

где

* – время выполнения проекта в месяцах (устанавливается в ТЗ и для этого проекта равно 2,6 месяца),
* – фонд времени в текущем месяце, который рассчитывается из учета общего числа дней в году, числа выходных и праздничных дней, рассчитываемого по формуле 18:

| , | (18) |
| --- | --- |

где

* – продолжительность рабочего дня,
* – общее число дней в году,
* – число выходных дней в году,
* – число праздничных дней в году.

Нерабочие праздничные дни в году устанавливается соответствующим законом [ссылка на статью 112 главы 18 трудового кодекса №197-ФЗ]. Подставив соответствующие значения рассчитаем среднемесячный фонд времени в формуле :

| , | (19) |
| --- | --- |

Подставив значения в формулу 17 получим фонд времени на период работы, определённых в техническом задании равным 428,1 часам.

### Определение количества исполнителей

Таким образом, при равномерном распределении работ для выполнения проекта в срок, установленный техническим заданием – 31 мая 2015 года – подставив в формулу 16 соответствующие значения и используя полные трудозатраты по всему проекту, получим среднее значение требуемого количества исполнителей в формуле 20:

| , | (20) |
| --- | --- |

### Календарный график выполнения работ

Для целей планирования и контроля работ проекта можно применить календарный ленточный график (диаграмма Ганта) – на оси Х 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 этапам определяется из формулы (21):

|  | (21) |
| --- | --- |

где

* – трудоёмкость i-й работы, чел.-часы;
* – трудоёмкость дополнительных работ для исполнителя, чел.-часы;
* – количество исполнителей для i-й работы.

В итоге была построена диаграмма Ганта указанных работ с учётом праздничных и выходных дней. Диаграмма представлена в приложении на рисунке А.1.

Длительность стадий проектных работ определяется по формуле 22:

| , | (22) |
| --- | --- |

где

* – продолжительность стадии, рабочие дни;
* – трудоёмкость этапа, чел/час;
* – доля дополнительных работ;
* – учитываемая продолжительность рабочего дня, часов/рабочий день, .

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

Таблица 10 – Оптимизированное количество исполнителей проектных работ

| № стадии | 1 | 2 | 3 | 4 | 5 |
| --- | --- | --- | --- | --- | --- |
| 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 [ссылка **Квалификационный справочник должностей руководителей, специалистов и других служащих** (утвержден постановлением Минтруда РФ от 21 августа 1998 г. N 37)], назначается следующим образом:

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

Разрабатываемое в ходе дипломного проектирования программное обеспечение выполняет функции управления входящим и исходящим потоками информации по специальным условиям, поэтому его можно отнести к функциональному назначению ПП «Управление технической подготовкой производства».

Средняя численность состава исполнителей при реализации проекта разработки и внедрения ПО определяется по формуле (23):

| , | (23) |
| --- | --- |

где

* - затраты труда на выполнение проекта (разработка и внедрение ПО),
* – фонд рабочего времени.

## Расчёт сметы затрат

Затраты на выполнение проекта состоят из прямых затрат (заработная плата исполнителям, затраты на закупку или аренду оборудования, затраты на организацию рабочих мест), и косвенных затрат (т.н. накладные расходы) вычисляются по формуле (29):

| , | (29) |
| --- | --- |

где

* + – заработная плата исполнителей;
  + – затраты на обеспечение необходимым оборудованием;
  + – затраты на организацию рабочих мест;
  + – накладные расходы.

Рассчитаем все составляющие затрат на разработку программного продукта.

### Затраты на выплату исполнителям

Затраты на выплату исполнителям заработной платы линейно связаны с трудоемкостью и определяются по формуле (30):

| , | (30) |
| --- | --- |

где

* + – основная заработная плата;
  + – дополнительная заработная плата;
  + – отчисление с заработной платы.

Расчёт основной заработной платы (оплаты труда непосредственных исполнителей) производится по формуле (31):

| , | (31) |
| --- | --- |

где

* + – число дней, отработанных исполнителем проекта;
  + – дневной оклад исполнителя.

При 8-и часовом рабочем дне он рассчитывается по формуле (32):

| , | (32) |
| --- | --- |

где

* + – месячный оклад;
  + – месячный фонд рабочего времени.

С учетом налога на доходы физических лиц размер оклада увеличивается, что отражено в формуле (33):

| , | (33) |
| --- | --- |

где

* + – «чистый» оклад;
  + – налог на доходы физических лиц в размере 13% .

В нашем проекте разработчиком ПО является программист C++, а проектировщиком – системный аналитик. Средние заработные платы по Москве были определены с помощью регулярных пресс-релизов электронного ресурса superjob.ru. Для программиста C++ (диапазон 3 в градации пресс-релиза) составляют 101500 рублей в месяц при полной рабочей неделе [1], а аналитика (диапазон 3) – 105000 рублей в месяц [2].

Итоговые заработные платы для работников указаны ниже (Таблица 11).

Следовательно, общие затраты на заработную плату исполнителям проекта составят 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 (34):

Таблица 11 – 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 |

|  | (34) |
| --- | --- |

Рассчитаем расходы на дополнительную заработную плату в формуле (35):

| руб. | (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, [3] указаны в таблице 12.

Таблица 12 – Ставки страховых взносов на 2014 год

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

Рассчитаем отчисления с заработной платы в формуле (36):

|  | (36) |
| --- | --- |

Таким образом, получим общие затраты на заработную плату в формуле (37):

|  | (37) |
| --- | --- |

### Суммарные затраты

Суммарные затраты вычисляются как сумма всех затрат по формуле (44):

|  | (44) |
| --- | --- |

Определим затраты на реализацию проекта в формуле (45):

| руб. | (45) |
| --- | --- |

## Вывод

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 данной работе реализован метод нахождения углов пространственной ориентации монокулярной камеры по полученным с нее изображениям с использованием точек схождения перспективы, предложенный Хуттунен, Пише [**Link1**].

Тестирования проводилось на наборе изображений YorkUrbanDb [**Link3**] Йоркского университета, состоящего из пейзажей города Торонто, а также внутренних помещений.

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. Средняя точность не превышает 2 градусов, что сравнимо с точностью ИНС потребительского класса.

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

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

# Библиография

Link1. A monocular camera gyroscope, Huttunen & Piché, 2012, <https://dspace.cc.tut.fi/dpub/bitstream/handle/123456789/21812/huttunen_piche_a_monocular_camera_gyroscope.pdf?sequence=1>

Link2. The York Urban Line Segment Database, <http://www.elderlab.yorku.ca/YorkUrbanDB/>

Link3. Технологии позиционирования в реальном времени, <http://www.rtlsnet.ru/technology/view/4>

Link4. Навигация мобильных роботов, <http://www.computer-museum.ru/frgnhist/robonav.htm>

Link5. Simultaneous localization and mapping, <http://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping>

Link6. Structure from motion, <http://en.wikipedia.org/wiki/Structure_from_motion>

Link7. SLAM For Dummies (A Tutorial Approach to Simultaneous Localization and Mapping), <http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf>

Link8. Bundle Adjustment — A Modern Synthesis, Bill Triggs, Philip McLauchlan, Richard Hartley and Andrew Fitzgibbon, <http://lear.inrialpes.fr/pubs/2000/TMHF00/Triggs-va99.pdf>

Link9. LSD: a Line Segment Detector, Rafael Grompone von Gioi, Jeremie Jakubowicz, Jean-Michel Morel, Gregory Randall, <http://www.ipol.im/pub/art/2012/gjmr-lsd/article.pdf>

Link10. The RANSAC (Random Sample Consensus) Algorithm, Robert B. Fisher, <http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/FISHER/RANSAC/>

Link11. Singular Value Decomposition (SVD), Principal Component Analysis (PCA), <http://www.math.nyu.edu/faculty/goodman/teaching/RPME/notes/Section3.pdf>

Link12. M. E. Antone and S. Teller. Automatic recovery of relative camera rotations for urban scenes. In IEEE Conference on Computer Vision and Pattern Recognition, 2000. <http://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TR-814.pdf>

Link13. Vanishing Point Detection For Architectural Photogrammetry, 1998, <http://www.lr.tudelft.nl/fileadmin/Faculteit/LR/Organisatie/Afdelingen_en_Leerstoelen/Afdeling_RS/Optical_and_Laser_Remote_Sensing/Research/Research_Fields/Heritage_and_Medical/Architectural_Photogrammetry/doc/Heuvelvanish.pdf>

Link14. Automated Architectural Acquisition from a Camera Undergoing Planar Motion, 2001, <http://www.robots.ox.ac.uk/~vgg/publications/2001/Montiel01/montiel01.pdf>

Link15. Photo Builder — 3D Models of Architectural Scenes from Uncalibrated Images, 1999, R. Cipolla, D. Robertson, E. Boyer, <http://mi.eng.cam.ac.uk/reports/svr-ftp/auto-pdf/cipolla_ICMCS99.pdf>

Link16. Using Vanishing Points To Correct Camera Rotation In Images, Gallagher, 2005, <http://chenlab.ece.cornell.edu/people/Andy/publications/Andy_files/rotation_crv2005.pdf>

Link17. Camera calibration from vanishing points in images of architectural scenes, 1999, R. Cipolla, V. Drummond, D. Robertson, <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.16.946&rep=rep1&type=pdf>

Link18. Using Vanishing Points for Camera Calibration, B. Caprile, V. Torre, 1990, <https://courses.engr.illinois.edu/cee598vsc/machineproblems/Using%20Vanishing%20Points%20for%20Camera%20Calibration.pdf>

Link19. Camera calibration from vanishing points in a vision system, B. He, Y. Li, 2008, <https://www.infona.pl/resource/bwmeta1.element.elsevier-80ac843d-605f-399d-8b06-d6a3ff69119f>

Link20. Camera calibration using two or three vanishing points, R. Orchidan, Joaquim Salvi, Mihaela Gordan, Bogdan Orza, <https://fedcsis.org/proceedings/2012/pliks/110.pdf>

Link21. Video compass, J. Kosecka, W. Zhang, 2002, <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.91.5186&rep=rep1&type=pdf>

Link22. Vision and Inertial Sensor Cooperation Using Gravity as a Vertical Reference, Jorge Lobo, Jorge Dias, 2003, <http://ap.isr.uc.pt/archive/17.pdf>

Link23. R. T. Collins and R. S. Weiss. Vanishing point calculation as a statistical inference on the unitsphere. In International Conference on Computer Vision, pages 400–403, 1990, <http://www.ri.cmu.edu/pub_files/pub2/collins_robert_1990_1/collins_robert_1990_1.pdf>

Link24. S. T. Barnard. Interpreting perspective images. Artificial

Intelligence, 21(4):435–462, 1983, <http://www.sri.com/sites/default/files/uploads/publications/pdf/672.pdf>

Link25. A. Almansa, A. Desolneux, and S. Vamech. Vanishing point detection without any a priori information. IEEE Transactions on Pattern Analysis and Machine Intelligence, 25(4):502–507, 2003, <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1190575&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F34%2F26687%2F01190575.pdf%3Farnumber%3D1190575>

Link26. J. M. Coughlan and A. L. Yuille. Manhattan world: compass

direction from a single image by bayesian inference. In

International Conference on Computer Vision, 1999. <http://www.stat.ucla.edu/~yuille/courses/Stat238-Winter12/ManhattanWorld.pdf>

Link27. J. M. Coughlan and A. L. Yuille. Manhattan world: Orientation and outlier detection by bayesian inference. Neural Computation, 15(5):1063–1088, 2003.

Link28. G. Schindler and F. Dellaert. Atlanta world: an expectation maximization framework for simultaneous low-level edge grouping and camera calibration in complex man-made environments. In IEEE Conference on Computer Vision and

Pattern Recognition, 2004.

Link29. P. Denis, J. H. Elder, and F. J. Estrada. Efficient Edge-Based methods for estimating manhattan frames in urban imagery. In European Conference on Computer Vision, pages 197– 210, 2008. <http://www.elderlab.yorku.ca/YorkUrbanDB/DenisElderEstradaECCV08.pdf>

Link30. Jean-Philippe Tardif. Non-Iterative Approach for Fast and Accurate Vanishing Point Detection, 2009. [http://www‑etud.iro.umontreal.ca/~tardifj/fichiers/Tardif\_ICCV2009.pdf](http://wwwetud.iro.umontreal.ca/~tardifj/fichiers/Tardif_ICCV2009.pdf)

Link31. Desolneux, A.; Moisan, L.; Morel, J.–M. From Gestalt Theory to Image Analysis. A Probabilistic Approach.2008, Hardcover. ISBN: 978-0-387-72635-9.

Link32. M. Zuliani, C. S. Kenney, and B. S. Manjunath. The Multi-RANSAC algorithm and its application to detect planar homographies. In International Conference on Image Processing, 2005.

Link33. Canny, J., A Computational Approach To Edge Detection, IEEE Trans. Pattern Analysis and Machine Intelligence, 8(6):679–698, 1986

Link34. Принцип работы системы навигации, [https://www.glonass‑iac.ru/guide/navfaq.php](https://www.glonassiac.ru/guide/navfaq.php)

Link35. Elena Tretyak, Olga Barinova, Pushmeet Kohli, Victor Lempitsky. Geometric Image Parsing in Man-Made Environments, 2011.

Link36. James Diebel. Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors. 2006. <http://www.swarthmore.edu/NatSci/mzucker1/e27/diebel2006attitude.pdf>

Link37. [http://docs.opencv.org/3.0‑alpha/doc/tutorials/introduction/windows\_install/windows\_install.html](http://docs.opencv.org/3.0alpha/doc/tutorials/introduction/windows_install/windows_install.html)

Link38. Средняя зарплата системного аналитика [http://www.it‑analytics.ru/analytics/trends/72773.html](http://www.itanalytics.ru/analytics/trends/72773.html)

Link39. Superjob.ru: средняя зарплата программиста C++ [http://www.it‑analytics.ru/analytics/trends/72773.html](http://www.itanalytics.ru/analytics/trends/72773.html)

Link40. Разработка ИНС, стоимость <http://www.warandpeace.ru/ru/news/view/98057/>

# Дополнительные иллюстрации

Диаграмма  — Диаграмма распределения точности обработки изображений «внутри помещения».

Диаграмма  — Диаграмма плотности распределения углов на изображениях «внутри помещения».

Диаграмма — Диаграмма распределения точности обработки изображений «вне помещения».

Диаграмма — Диаграмма плотности распределения углов на изображениях «вне помещения».

1. Диаграмма Ганта выполняемых работ

Таблица 12 – Диаграмма Ганта выполняемых работ

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | Название этапа | Продолжительность,  раб. дни | Исполнители | | Календарные дни | | | | | | | | | | | |
| Категория | Число | 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 |
| Количество рабочих дней | | | | | | | | | | | |
| 4 | 5 | 5 | 5 | 5 | 5 | 5 | 4 | 4 | 4 | 5 | 5 |
| 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 |  |  |  |  |  |  |  |  |  |  |