Ориентация монокулярной камеры с использованием точек схождения перспективы

Кудеров П.В.

ИУ9-121

**Оглавление**

[Введение 3](#_Toc420922595)

[1 Аналитическая часть 5](#_Toc420922596)

[1.1 Обзор систем навигации 5](#_Toc420922597)

[1.1.1 Инерциальные системы 5](#_Toc420922598)

[1.1.2 Спутниковые системы 5](#_Toc420922599)

[1.1.3 Другие системы локального позиционирования 6](#_Toc420922600)

[1.2 Системы позиционирования с использованием методов компьютерного зрения 6](#_Toc420922601)

[1.2.1 Методы SLAM и SfM 6](#_Toc420922602)

[1.2.2 Методы на основе исследования точек схождения перспективы 7](#_Toc420922603)

[2 Научно-исследовательская часть 9](#_Toc420922604)

[2.1 Геометрия перспективных изображений 13](#_Toc420922605)

[2.2 Обнаружение ТСП 20](#_Toc420922606)

[3 Проектно-конструкторская часть 23](#_Toc420922607)

[4 Технологическая часть 24](#_Toc420922608)

[4.1 Методология разработки и используемые средства 24](#_Toc420922609)

[4.2 Руководство пользователя 24](#_Toc420922610)

[4.2.1 Приложение Gyrocam 24](#_Toc420922611)

[5 Организационно-экономическая часть 29](#_Toc420922612)

[5.1 Введение 29](#_Toc420922613)

[5.2 Организация и планирование процесса разработки программы 30](#_Toc420922614)

[5.2.1 Техническое задание 30](#_Toc420922615)

[5.2.2 Расчёт стоимости проекта 30](#_Toc420922616)

[5.2.3 Затраты на выплату исполнителям 33](#_Toc420922617)

[5.2.4 Определение количества исполнителей 42](#_Toc420922618)

[5.2.5 Календарный график выполнения работ 43](#_Toc420922619)

[5.3 Расчёт сметы затрат 45](#_Toc420922620)

[5.3.1 Затраты на выплату исполнителям 46](#_Toc420922621)

[5.3.2 Суммарные затраты 49](#_Toc420922622)

[5.4 Цена продукта 50](#_Toc420922623)

[5.5 Вывод 51](#_Toc420922624)

[Заключение 52](#_Toc420922625)

[Приложение 1. Диаграмма Ганта выполняемых работ 54](#_Toc420922626)

# 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 определения трехмерной ориентации монокулярной камеры с использованием точек схождения перспективы (ТСП), обнаруженных на изображениях.

Создатели метода выделяют следующий набор его достоинств (***ниже почти дословная цитата из******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 [**Link1**] в условия городской застройки. Для тестирования результатов выбран набор тестовых данных YorkUrbanDb [**Link3**], 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 решений… 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

[*image: nav systems relation*]

Среди популярных применяемых в автономной робототехнике решений – использование инерциальных систем навигации (ИНС). Подобные системы содержат акселерометры 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.

### Спутниковые системы

Не имеют дрейфа глобальные (спутниковые) системы навигации (GPS, GLONASS, GALILEO и другие их аналоги). Для существующих систем данного класса характерна погрешность позиционирования порядка 5-15 метров (***данные википедии***). 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 (***практически*** ***цитата из ненаучной статьи***). 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 технологий построения трехмерной карты окружающей среды на основе данных эхолокации инфракрасными или ультразвуковыми датчиками.

Набирает популярность использование методов компьютерного зрения. Видеодатчики имеют маленький размер, энергопотребление, цену, поэтому есть интерес использовать их в системах позиционирования и навигации. Для тестирования методов можно использовать даже камеры мобильных телефонов. Системы позиционирования 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.

## Системы позиционирования с использованием методов компьютерного зрения

### Методы SLAM и SfM

Среди распространенных – метод одновременной навигации и картирования (simultaneous localization and mapping, SLAM), а также метод определения структуры объекта в процессе движения (structure from motion, SfM). Оба этих метода в той или иной степени пытаются построить 3D модель окружающей среды, относительно которой происходит движение камеры.

[*image: slam & sfm examples*]



Рисунок – Пример результата работы SLAM метода

\* SLAM – используются Particle filters / Sequential Monte Carlo(SMC) или расширенный фильтр Калмана, bundle adjustment (корректировка узлов?)

\* SfM (Structure from motion) - видимо, то же, что и 3D reconstruction from 2D images, распространен алгоритм Scale-invariant feature transform (SIFT), другой распространенный алгоритм - Speeded Up Robust Features (SURF).

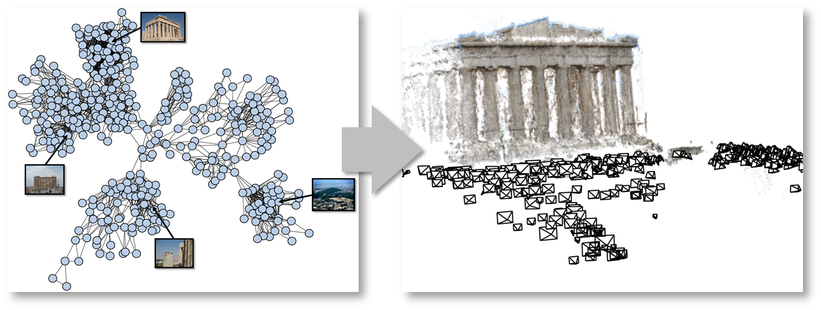


Рисунок – Пример результата работы SfM метода

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

### Методы на основе исследования точек схождения перспективы

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

[*image: mahattan world example, indoor, city outdoor, non-city outdoor*]

Данный метод был протестирован создателями метода внутри помещений. В таких условиях хорошо различимы на изображении углы между стенами, полом и потолком. Линии, образованные данными углами, образуют ортогональную систему осей координат, что позволяет по проекциям этих линий вычислить углы ориентации камеры в пространстве с точностью, сравнимой с ИНС потребительского класса.

Плюсы конкретно данного метода (***ниже почти дословная цитата из******Link1***):

* + в условиях "Manhattan World" изобилие линейных объектов правильной геометрической формы позволяет достигнуть высокой точности
  + число интересующих нас ТСП ограничено <= 3
  + ТСП не зависят от положения камеры - только от ее ориентации
  + робастность относительно случайных нестационарных объектов, попадающих в кадр (люди, транспортные средства и т.п.)

Методы на основе определения ТСП позволяют решать задачи:

* + навигации автономных транспортных средств, роботов
  + 3D реконструкции окружающей среды
  + калибровки камер и коррекции изображений

Метод был протестирован на наборе изображений YorkUrbanDb [**Link3**], созданного на базе York University Centre for Vision Research, а также на случайных изображениях, взятых из интернета или сделанных вручную.

[*image: our result image*]



Набор YorkUrbanDb состоит из 47 изображений внутри помещений и 55 изображений городских сцен с вычисленными значениями ТСП.

Существует целый ряд работ, посвященных вычислению ТСП на снимках городских сцен и внутри помещений, то есть в условиях так называемого «Manhattan World» [**Link2**]..

Промежуточная цель работы состоит в исследовании применимости метода Хуттунена и Пише в условиях съемок внутренних помещений и городских пейзажей, обнаружение слабых мест и ознакомление с аналогичными работами по данной тематике, предложение вариантов улучшения метода, нахождение удобного способа тестирования (?).

Конечная цель – создание библиотеки / приложения, которое позволяет эффективно и с точностью, сравнимой с гироскопами потребительского класса, вычислять углы наклона и поворота камеры по входному потоку изображений в условиях «Manhattan World assumption» в реальном времени..

### Краткий обзор аналогичных работ

...

# Научно-исследовательская часть

(Очень грубое разделение на шаги) Используемый в данной работе метод нахождения углов поворота и наклона состоит из следующих шагов:

* регистрация и чтение изображения,
* выделение сегментов линий на изображении
* выделение наилучших кластеров СЛ, сходящихся в одной точке (или бесконечности)
* перевод координат сегментов линий в нормализованные координаты
* вычисление вектора направления ТСП на основе полученных кластеров СЛ путем приближенного решения задачи минимизации Ax = 0
* ортогонализация матрицы, составленной из векторов направлений ТСП
* получение углов Эйлера в нотации (1, 2, 3) из матрицы поворота

## Геометрия перспективных изображений

Проективное преобразование и однородные координаты, модель булавочного отверстия, ТСП

Теория:

1. Геометрия ТСП

Теория точек схождения перспективы (ТСП) рассматривается обычно в терминах проективной геометрии, изучающей геометрические свойства, являющихся инвариантными 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.

1.1. Проективное преобразование и однородные координаты, модель булавочного отверстия, ТСП

В проективной геометрии точки представлены в однородных координатах. Например, рассмотрим точку в двухмерном евклидовом пространстве (x, y) @ R2. Чтобы представить ее на проективной плоскости, необходимо лишь добавить третью компоненту z, равную 1, т.е. (x, y, 1) @ P2. Более того, для любого ненулевого a (ax, ay, a) = (x, y, 1). В случае нулевого a координаты точки вырождаются в точку (0, 0, 0), которая не включена в P2.

Рассмотрим уравнение прямой на двухмерной евклидовой плоскости:

ax + by + c = 0

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

ax + by + c\*1 = aX + bY + cZ = 0

Таким образом мы определили все точки (x, y, 1) = (X / Z, Y / Z, 1) = (X, Y, Z) проективного пространства, лежащие на прямой. Заметим также, что aX + bY + cZ = l \* p = 0, где l = (a, b, c), p = (X, Y, Z), a \* - скалярное произведение. 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:

\* проверка принадлежности точки прямой incidence(p1, p2) = p1 \* p2

\* прямая через две точки ltp(p1, p2) = p1 x p2, где x - векторное произведение

\* точка пересечения двух прямых intersection(l1, l2) = l1 x l2

Чтобы вернуться из P2 обратно в R2, достаточно поделить координаты точки на z-координату, то есть (X, Y, Z) = (X / Z, Y / Z, 1) = (x, y, 1) @ P2 ~ (x, y) @ R2. Из данной процедуры сразу видно, что P2 ( R2, так как содержит элементы с z = 0. Эти точки составляют довольно важно подмножество P2 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 идеальной прямой или прямой в бесконечности.

Также заметим, что точкам p(X, Y, Z) = (x, y, 1) @ P2 можно поставить в соответствие прямую, проходящую через начало координат и точку p`(x, y, 1) @ R3 с выколотой точкой (0, 0, 0). Таким же образом линия l(a, b, c) на проективной плоскости может быть визуализирована в R3 плоскостью, образованной началом координат и перпендикуляром к l`(a, b, c). Тогда точкам с координатами (x, y, 1) в R3 соответствует плоскость Z = 1, идеальным точкам соответствуют точки на Z = 0, а идеальной прямой сама плоскость Z = 0.

Данная связь между R3 и P2 может быть легко продолжена до связи между P3 (добавлением к точкам R3 четвертой координаты 1) и P2. Такая связь очень хорошо подходит для описания преобразования проецирования трехмерной сцены на двухмерную плоскость изображения.

Также заметим, что точкам p(X, Y, Z) = (x, y, 1) @ P2 можно поставить в соответствие прямую, проходящую через начало координат и точку p`(x, y, 1) @ R3 с выколотой точкой (0, 0, 0). Таким же образом линия l(a, b, c) на проективной плоскости может быть визуализирована в R3 плоскостью, образованной началом координат и перпендикуляром к l`(a, b, c). Тогда точкам с координатами (x, y, 1) в R3 соответствует плоскость Z = 1, идеальным точкам соответствуют точки на Z = 0, а идеальной прямой сама плоскость Z = 0.

Данная связь между R3 и P2 может быть легко продолжена до связи между P3 (добавлением к точкам R3 четвертой координаты 1) и P2. Такая связь очень хорошо подходит для описания преобразования проецирования трехмерной сцены на двухмерную плоскость изображения.

Рассмотрим точку в R3 p(X, Y, Z). Для того, чтобы прибавить к ней вектор t(Tx, Ty, Tz), мы можем воспользоваться следующим матричным выражением:

[p + t] = [ E t] \* [p]

[1 ] [ 0 1] [1]

Похожим образом выражается поворот точки p и умножение каждой из его координат на независимый коэффициент d(dx, dy, dz):

[p'] = [R 0] \* [p]

[1 ] [0 1] [1]

[dx \* X] [dx 0 0 0]

[dy \* Y] = [0 dy 0 0] \* [p]

[dz \* Z] [0 0 dz 0] [1]

[1 ] [0 0 0 1]

Можно заметить, что в данных выражениях точка p @ R3 представлена в нормализованных координатах P3 (X, Y, Z, 1). Также очевидно, что данные выражения остаются верными, если и для точкек вида alpha \* p = (alpha X, alpha Y, alpha Z, alpha):

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

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

\* перенос на вектор t оставляет идеальную точку на месте

\* поворот действует на идеальную точку абсолютно также, как и на конечную

\* масштабирование на вектор d(dx, dy, dz) действует аналогично действию на конечную точку

Теперь рассмотрим связь между системами координат камеры и мировой системой координат. Пусть в мировой системе координаты камеры представлены точкой t, а матрица поворота R связывает соответствующие оси систем, тогда выражения связи имеет вид:

pw -> R(pw - t) = pc

, где pw - точка в мировой системе, а pc - в координатах системы камеры.

pc = Rpw - Rt

Если же воспользоваться нормализованными координатами, получится следующее:

[pc] = [R -Rt] \* [pw] = [R 0] \* [I -t] \* [pw]

[1 ] [0 1] [1 ] [0 1] [0 1] [ 1]

Матрица M, определенная как:

M = [R 0] \* [I -t]

[0 1] [0 1]

, задает матрицу перехода между системами координат. Матрицы R и t задают внешние (extrinsic) параметры камеры - ориентацию и позицию - в мировых координатах.

Одна из самых простых и обычно используемых моделей для конечной проективной камеры - модель булавочного отверстия или модель камеры обскуры (pinhole camera), в которой точки p(X, Y, Z) @ R3 проецируются на двухмерную плоскость по правилу:

(x, y) = (fX / Z, fY / Z)

или в нормализованных координатах:

(x, y, 1) = (f X/Z, fY / Z) = |при Z != 0| = (X, Y, Z / f) = (fX, fY, Z)

Данное правило может быть записано в виде матрицы проекции:

[fX] [f 0 0 0] [X]

[fY] = [0 f 0 0] = [Y]

[Z ] [0 0 1 0] [Z]

[1]

Очевидно также, что вектор [0 0 0 1]T - является нуль-вектором нуль-пространства данного преобразования.

Работая с реальными камерами и точками на полученных с них изображениях, удобнее иметь дело с координатами, выраженными в пикселах нежели, например, в миллиметрах. Перевод координат требует информации о линейных размерах пиксела (например, в миллиметрах) и координат главной точки (principal point), которой соответствует центр изображения (точка пересечения оптической оси камеры с плоскостью изображения), так как в общем случае она может не совсем точно совпадать с центром матрицы камеры (более того, довольно часто центр координат в пикселах определяется одним из углов изображения).

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

K = [ f/mx s px ]

[ 0 f/my py ]

[ 0 0 1 ]

, где

\* f - фокусное расстояние камеры в некоторой единице длины (обычно в мм или дюймах),

\* mx, my - линейные размеры пиксела, выраженные в той же единице длины, что и f. Таким образом f / mx и f / my имеют размерность пикселов (мм / (мм / px) = px),

\* s - коэффициент ассиметрии камеры

\* px, py - координаты главной точки в пикселах.

Большинство цифровых камер на основе приборов с зарядовой связью (ПЗС) имеют квадратные пикселы (mx = my), нулевую ассиметрию (s = 0) и главную точку, расположенную близко к центру изображения.

! возможно стоит сказать про дисторсию, что она есть, но в данной работе мы ей принебрегаем

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

F: P3 -> P2, F(p) = p' = KR[I | -t] p

Рассмотрим прямую в P3, заданную как X(alpha) = A + alpha \* D, где А - точка на этой прямой, D(d, 0) - направляющий вектор и d @ R3, alpha @ R. Проекция этой прямой:

x(alpha) = P(X(alpha)) = P(A) + alpha \* P(D) = P(A) + alpha \* KR[I | -t]D = P(A) + alpha \* KRd

, так как [I | -t]D = d.

ТСП vp @ P2, соответствующая направлению d, является предельной точкой для проекции линии x(alpha) при alpha -> inf:

vp = lim x(alpha) = lim( P(A) + alpha \* KRd) = KRd

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

vp = Kd

Из полученного результата можно заключить, что:

\* ТСП vp не зависит от положения t камеры

\* существует взаимно однозначное отношение между vp 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 / перпендикулярно друг другу. Например, столы, полки, окна или витрины, пол, стены и потолок - внутри помещений, дома, дороги и разметка на них, окна домов - вне помещений.

## Обнаружение ТСП

### Обнаружение СЛ

Вкратце преобразование Хо (Хаф, Hough Transform) и анализ связанных компонент ориентаций градиента изображения, возможно про Canny детектор, кратко про LSD (pdf)

Выбрали анализ ... почему?

Lsd - O(img.size()), but slower than Hough

Первым этапом является выделение сегментов линий на изображении. Для решения данной задачи в оригинальной статье предлагается использовать метод Джиои и ... (Link4). (проверить, что написано во введение самой статьи Джиои) Это довольно новый алгоритм, отличающийся высокой скоростью за счет линейной зависимости сложности от размеров изображения по сравнению с другими алгоритмами, основанных на анализе связанных компонент градиента изображения. По быстродействию он уступает алгоритмам, основанным на преобразовании Хафа (Hough), но позволяет достичь более высокого качества.

### Кластеризация СЛ

RANSAC, можно (нужно! но, возможно, не здесь, а дальше в "что можно улучшить") заикнуться про J-Linkage (введение в статье pdf - сравниваются разные способы кластеризации)

RANSAC выбрали, потому что популярное решение.

Расписать функцию расстояния d(vp,l), как в статье, как делаю я.

Расписать как выбирается число итераций алгоритма (вывод формулы)

На данном этапе выработана грубая оценка ТСП

Далее, я провожу кластеризацию сегментов линий, используя адаптивный алгоритм RANSAC. На каждом прогоне алгоритма вычисляется самый большой оставшийся кластер сегментов, линии которых пересекаются в одной точке с некоторой допустимой погрешностью. Полученный кластер объявляется соответствующим некоторой ТСП. Всего производится 3 запуска алгоритма.

Алгоритм состоит из последовательности итераций, состоящих из следующих шагов:

\* случайным образом выбирается пара сегментов линий

\* вычисляется точка их пересечения, которая объявляется потенциальной ТСП

\* далее проводится каждого сегмента линии на предмет принадлежности его потенциальной ТСП путем вычисления функция расстояния до ТСП по формуле:

Point3d l = lineThroughPoints(segment.middle, vp);

double d = incidence(l, segment.from) / norm12(l);

return abs(d) <= distanceEpsilon

&& abs(asin(d / norm(segment.middle - segment.from))) <= angleEpsilon;

, где

static const double distanceEpsilon = 2; // 2 pixel

static const double angleEpsilon = 0.04; // ~1% pi or ~2 degree

Таким образом для имеющейся потенциальной ТСП определяется множество соответствующих ей сегментов. Назовем содержащиеся в нем сегменты внутренними, а все остальные - внешними по отношению к данной vp. Обозначим также их отношение как r. Чем выше число r, тем более подходящей считается ТСП.

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

P(k) = (1 - r^2)^k

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

p >= 1 - P(k)

=> P(k) >= 1 - p

=> (1 - r^2)^k >= 1 - p

логарифмируем обе стороны:

=> k >= log(1 - p) / log (1 - r^2)

Истинное значение r неизвестно, но его можно аппроксимировать снизу r', соответствующим наилучшей из найденных за текущие k' итераций ТСП.

После каждого запуска алгоритма RANSAC внутренние сегменты для найденной ТСП удаляются из выборки и в последующих запусках не участвуют. В итоге после 3х последовательных запусков алгоритма мы имеем 3 кластера сегментов линий, каждому из которых соответствует довольно грубая оценка ТСП.

### Уточнение ТСП

Описать математически текущую ситуацию - пучок линий сходится примерно в одной точке. Какое уравнение связывает их в идеале.

Значит решаем Ax = 0 приближенно. Что делать, если система не переопределена (у меня сейчас просто упадет программа!)?

Какие существуют способы приближенного решения данной задачи? LSQM, SVD, smth else

Почему мы не решаем ее тут, а переходим в нормализованные координаты изображения? Что это такое вообще, зачем? В честь кого это делается? [Cipolla]

Получили нулевые направления, что дальше?

Так как координаты сегментов линий имеют приблизительный характер, ТСП, выбранная на основе случайной пары из этого множества, не "учитывает" голоса остальных сегментов. Поэтому производится уточнение координат ТСП через минимизацию функционала, учитывающего все сегменты соответствующего ТСП множеста.

Рассмотрим сегменты одного из множеств. Линии, заданные ими, пересекаются в некоторой точке vp, т.е.

для каждого segment @ Cluster: ltp(segment.line, vp) = segment.line \* vp = 0

Или для всего множества:

[segment1.line .. segmentN.line]T \* vp = 0

Таким образом имеется переопределенная система уравнений. Так как данное уравнение имеет приблизительный характер, на самом деле мы хотим найти такую нетривиальную точку vp, которая минимизирует функционал в левой части, а именно:

vp = argmin(LT \* vp)

Существуют разные методы решения данной задачи. В данной работе мы выбрали метод, использующий сингулярное разложение матрицы LT. В библиотеке opencv существует класс cv::SVD, в котором реализован функционал сингулярного разложения матриц. В частности существует статический метод cv::SVD::SolveZ, который решает задачу минимизации функционала (\*). Нами было принято решение воспользоваться данным методом.

Смысл метода заключается в следующем: при сингулярном разложении некоторой матрицы A –

....

### Определение ориентации по ТСП

Есть матрица D = [d1, d2, d3]. Если только две приблизительно ортогональны (или 3ей не нашли вообще), 3ю вычисляем на основе их через векторное произведение.

Объясняю, что D скорее всего не ортогональна => можно ортогонализовать. Как? SVD! D' = u \* vt

Приводим матрицу нулевых направлений

Матрица относительного поворота R = D2 D1t

At = A-1, т.к. ортогональная => alpha(R) = alpha(D2) + alpha(D1t) = alpha(D2) - alpha(D1)

# Проектно-конструкторская часть

Приложение состоит из следующих функциональных блоков:

* организация конфигурации запуска приложения, обеспечение ввода и вывода данных, управление тестированием
* чтение изображения и выделение сегментов линий
* кластеризация сегментов линий
* вычисление и уточнение точек схождения перспективы
* вычисление углов поворота камеры

Входные данные алгоритма - изображение с условием "Manhattan World".

Первым этапом является выделение сегментов линий на изображении. Для решения данной задачи в оригинальной статье предлагается использовать метод Джиои и ... (Link4). (проверить, что написано во введение самой статьи Джиои) Это довольно новый алгоритм, отличающийся высокой скоростью за счет линейной зависимости сложности от размеров изображения по сравнению с другими алгоритмами, основанных на анализе связанных компонент градиента изображения. По быстродействию он уступает алгоритмам, основанным на преобразовании Хафа (Hough), но позволяет достичь более высокого качества.

В нашем приложении мы использовали реализацию алгоритма Джиои в библиотеке opencv версии 3.0.0. Ему отвечает класс cv::LineSegmentDetector (LSD), который принимает на вход изображение в градациях серого (в opencv тип CV\_8UC1), и различные параметры настройки алгоритма. В своем приложении в качестве параметров настройки я использовал рекомендованные по умолчанию.

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

\* координаты концов в пикселах в виде четверки целых чисел (тип данных opencv Vec4i)

\* ширина линии

\* точность, с которой он найден

\* число ложных срабатываний (number of false alarms) в области сегмента линии в виде логарифмической шкалы качества детектирования

Получив результат работы детектора сегментов линий, я провожу фильтрацию сегментов по длине, отбрасывая слишком короткие меньше 20 символов. Во-первых, это позволяет значительно ускорить работу на последующих этапах. Во-вторых, мною было замечено, что короткие сегменты чаще относятся к ошибочным направлениям (не к искомым ТСП). Те же, что относятся к искомым ТСП за счет своей длины ухудшают точность вычислений, т.к. погрешность детектора в обнаружении одного из концов отрезка в 1 пиксел ведет к достаточно большой итоговой угловой погрешности.

Далее я провожу подготовку структур данных к последующим этапам работы алгоритма, создавая на основе каждого отрезка, представленного четверкой целых чисел, объект структуры LineSegment, вычисляю и сохраняю следующие поля:

\* оригинальную четверку координат концов, полученных в виде объекта структуры Vec4i

\* координаты точек концов отрезка в нормализованных координатах и удобном формате типа cv::Point3d

\* середину отрезка в нормализованных координатах

\* уравнение линии, которую задает сегмент, в нормализованных координатах. Уравнение вычисляется как векторное произведение точек концов отрезка

# Технологическая часть

## Методология разработки и используемые средства

## Руководство пользователя

### Приложение Gyrocam

Результатом работы является Win32 консольное приложение, реализующее алгоритм нахождения точек схождения перспективы (ТСП) на изображении, а также векторов единичных направлений, соответствующих им.

Для использования данного приложения требуется установленная на компьютере пользователя библиотека компьютерного зрения opencv версии 3.0.0. Описание процесса установки и необходимые файлы можно найти по адресу [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).

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 cv::SVD::solveZ
* 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** 1).

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.

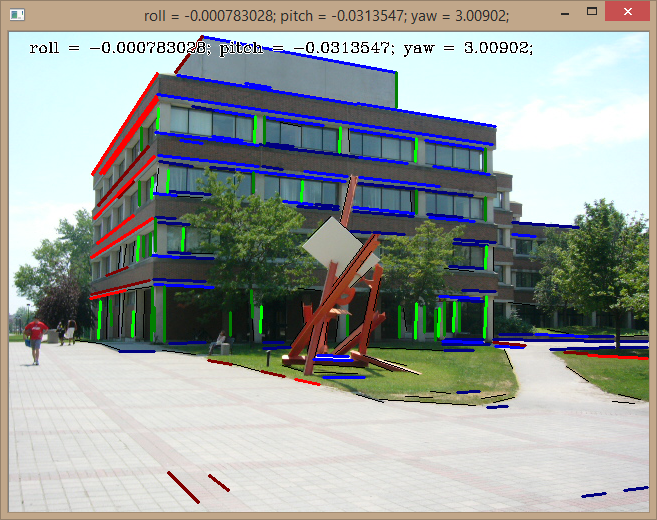


Рисунок – Окно с обработанным изображением, поверх которого наложены сегменты линий, соответствующие найденным ТСП.

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** 2). 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"

### Режим тестирования набора изображений

Режим тестирования набора изображения задатеся флагом ‑yorkUrbanDb.

… Пример строки аргументов:

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

## Тестирование приложения

Описанное консольное приложение было протестировано на наборе изображений YorkUrbanDb (The York Urban Line Segment Database). Набор состоит из 47 изображений внутри помещений и 55 изображений городских сцен. Для каждого изображения из базы приведены данные по точкам схождения перспективы, что позволяет оценить уровень точности работы приложения. Описание и сам набор изображений можно найти на странице базы <http://www.elderlab.yorku.ca/YorkUrbanDB/>.

Также может представлять интерес для тестирования набор изображений в составе так называемой Eurasian Cities Database. Описание и сам архив базы для скачивания доступен на <http://graphics.cs.msu.ru/en/research/projects/msr/geometry>.

## Результаты тестирования

# 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) |
| --- | --- |

График распределения затрат по категориям представлен на рисунке 9.

## Цена продукта

Если ПП рассматривается и создаётся как продукция производственно-технического назначения, допускает многократное тиражирование и отчуждение от непосредственных разработчиков, то её цена определяется по формуле (46):

| , | (46) |
| --- | --- |

где

* – затраты на разработку ПО;
* – коэффициент учёта затрат на изготовление опытного образца ПП как продукта производственно-технического назначения (К = 1,1…1,2);
* – нормативная прибыль, рассчитываемая по формуле (47):

Рисунок 9 – Распределение затрат на реализацию проекта

| , | (47) |
| --- | --- |

где

* – материальные затраты (в нашем случае, затраты на оборудование);
* – норматив рентабельности, в %.

Минимальное значение рентабельности должно быть выше средней процентной ставки по депозитным вкладам в коммерческом банке. Возьмём за точку отсчёта .

Вычислим нормативную прибыль в формуле (48):

| руб. | (48) |
| --- | --- |

Коэффициент учёта затрат на изготовление опытного образца ПП возьмём минимальный .

Следовательно, цена разрабатываемого ПП получим в формуле (49):

| руб. | (49) |
| --- | --- |

## 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. Проведено исследование работы метода на наборе случайных изображениях городской и сельской местности, взятых из сети интернет или сделанных вручную.

В условиях внутренних помещений метод показал достаточно высокий уровень точности. Произведена попытка перенести метод в городские условия. На тестовых данных результаты оказались хорошими, но несколько хуже, чем во внутренних помещениях.

Анализ проблем, полученных при тестировании.

Библиография

Link1. Huttunen & Piché, 2012

Link2. Coughlan & Yuille, 1999, 2003

Link3. The York Urban Line Segment Database

Link4. Средняя зарплата системного аналитика [http://www.it‑analytics.ru/analytics/trends/72773.html](http://www.itanalytics.ru/analytics/trends/72773.html)

Link5. Superjob.ru: средняя зарплата программиста C++ [http://www.it‑analytics.ru/analytics/trends/72773.html](http://www.itanalytics.ru/analytics/trends/72773.html)

Link6. Разработка ИНС, стоимость http://www.warandpeace.ru/ru/news/view/98057/

1. Диаграмма Ганта выполняемых работ

Таблица 1 – 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 |  |  |  |  |  |  |  |  |  |  |