**Введение**

Проблема навигации внутри помещений является важной в существующей экосистеме. Многие задачи нашего времени исполняются автономными техническими средствами – роботами. Для выполнения возложенной людьми работы необходимо обеспечить их средствами, чтобы они могли избежать всех препятствий, с которыми они могут столкнуться, проехать через узкие пространства и т.д. Все это становится возможно с применением камер, датчиков – необходимо сделать робота больше похожим на человека.

Навигация робота означает его способность определять собственную позицию в своей системе отсчета, а затем планировать путь к некоторому расположению цели. В отличие от человека робот должен справляться без карты окружения, он должен быть способен воспроизвести её представление.

Навигация может быть определена как комбинация трех основных положений:

* Ориентирование в пространстве
* Планирование пути
* Построение карты и её интерпретация

Ориентирование в пространстве — значит способность робота установить свое положение и ориентацию в своей системе отсчета. Планирование пути фактически является продолжением ориентирования в пространстве, оно требует определения текущего положения робота и местоположения цели во внешней системе отсчета, следовательно, роботу необходимо сопоставить собственную систему отсчета и окружения. Построение карты может быть выполнено в процессе выполнения движения по построенному ранее маршруту, для использования в дальнейшем.

В данной работе будет решена проблема ориентирования робота в пространстве с использованием камеры, а также будет разработана программа позволяющая обрабатывать видеофайл с камеры робота.

**Глава 1. Обзор существующих аналогов**

Существует множество систем, которые позволяют ориентироваться автономному транспортному средству в пространстве. Технологии используются как вне, так и внутри помещений.

* 1. Система предупреждения о сходе с полосы

Система предупреждения о сходе с полосы чаще всего является механизмом, предназначенным для предупреждения водителя, когда автомобиль начинает выдвигаться из своей полосы (при выключенном сигнале указателя поворота) на автострадах и магистралях, но также она может быть использована для автономной навигации беспилотных транспортных средств. Такая система предназначена для минимизации несчастных случаев путем устранения основных причин аварий: ошибки, рассеянность и сонливость водителя.

Система представляет собой камеру, которая установлена на верхней части лобового стекла автомобиля, часто вмонтированная в зеркало заднего вида. Изображение пространства перед автомобилем передается алгоритму, который анализирует его для выявления прямых или пунктирных линий - разметки дорожного полотна. Алгоритм использует преобразование Хафа и детектор границ Канни для распознавания дорожной линии. В режиме реального времени изображение с камеры поступает обработчику.

При предварительной обработки на изображении выделяются, а затем удаляются шум. Посредством этого изображение делается более четким. Затем картинка переводится в оттенки серого и на неё накладывается детектор границ Канни, который выделяет видимые грани на изображении. После этого применяется преобразование Хафа, обнаруживающее линии на изображении.[1]

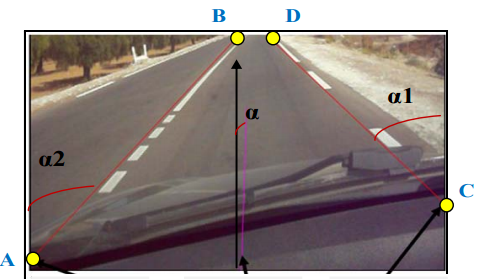


Рисунок 1.1 – Обнаруженные линии разметки

Найденные и отфильтрованные по разметке линии напоминают конус, который смыкается приблизительно в середине изображения, подсчитываются углы отклонения этих линий и рассчитывается угол поворота руля относительно углов отклонения по формуле 1.1

, (1.1)

где угол наклона линий.

Также существует другой алгоритм контроля полосы, основанный на разделении проезжей части от окружения.

Предварительная обработка таких изображений также начинается с перевода цветного RGB изображения в оттенки серого с удалением шума и получения его более четким. На втором этапе выискиваются такие области картинки, которые похожи на дорогу для того, чтобы выделить необходимую для работы часть. Эта область выделяется на основе специфических геометрических данных проезжей части на изображении. Следующим шагом производится фильтрация дороги от остальной части изображения и окрашивает её в красный цвет.

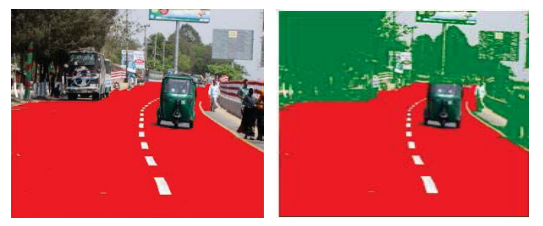


Рисунок 1.2 – Выделение проезжей части

Алгоритм проходит изображение попиксельно слева направо, и справа налево одновременно, окрашивая все остальные пиксели в зеленый цвет. Когда встречается красный цвет алгоритм переходит на следующую строку и выполняется снова до конца изображения.

Подбирая различные параметры фильтрации можно добиться устранения теней и других объектов, которые могут помешать обнаружению проезжей части.[2]

В системах помощника водителю, руль обычно просто начинает вибрировать, привлекая внимание, но в автономных системах эта система контролирует положение автомобиля на дороге, позволяя избегать аварий без помощи человека.

* 1. Роботы-погрузчики на складах

Самым известной таких роботов является компания Amazon, разработавшая систему Kiva. При расположении заказа в базе данных, программа находит ближайшее автоматизированное транспортное средство (бота) к точке загрузки и направляет его туда. Мобильные роботы перемещаются по территории склада посредством компьютеризированных наклеек на полу. Каждый робот имеет датчик, который предотвращает его от столкновения с другими. Прибывая на точку погрузчик забирает товар и направляет его нужному оператору для дальнейшей обработки. Такая система освобождает от работы многих людей и позволяет сократить расходы на содержание складов и персонала.

Роботы имеют небольшие размеры, порядка 2 х 2,5 фута и весят около 250 килограмм. Они оснащены подъемным механизмом, который позволяет поднять нужную полку с товарами. Работают они от свинцово-кислотных аккумуляторов, которые заряжаются время от времени в течение дня. Цикл жизни таких батарей варьируется от 1,5 до 2 лет непрерывной работы.

Передвигаются роботы по сетке на полу, которая помечена метками для навигации. Они размещены через каждые 2-3 метра, что посредством сетки позволяет эффективно ориентироваться роботам на всей площади склада. На роботе установлено две камеры, одна направлена вверх, другая вниз. Они и отслеживают те самые наклейки, которыми и осуществляется навигация. Также камеры установлены на площади склада, которые считывают наклейки непосредственно на самих роботах, что позволяет отслеживать их местоположение на территории.



Рисунок 1.3 – Робот-погрузчик компании Amazon

Каждый робот оснащен ультразвуковыми датчиками, которые оповещают о препятствиях на пути и не позволяет ему двигаться дальше. Эти датчики действуют на случаи, если перед роботом окажется человек или товар упадет с него.

Огромным преимуществом является то, что роботы могут работать в темноте, что также позволяет сократить расходы на электроэнергию и кондиционирование.

Важным аспектом в работе всей системы является программное обеспечение центрального компьютера. Оно фактически отвечает за логичное и бесперебойное движение всех роботов по территории склада, является мозгом всей системы. ПО включает в себя довольно сложный набор алгоритмов, которые необходимы, чтобы свести к минимуму время работы операторов одновременно гарантируя, что количество работающих роботов будет также сводится к минимуму. Программа автоматически решает куда и как именно двигаться роботу, чтобы покрыть максимально большее пространство склада. Время решения компьютера гораздо быстрее, чем то время, которое затратит оператор на поиск оптимального пути.[3]

Все роботы управляются посредством единой сети Wi-Fi. Они передают данные о пройденных метках на центральный компьютер, в то время как он отправляет дальнейшие указания роботам по передвижению. Как правило для этих операций хватает обычного роутера, которые используются в домашних условиях.

Такая система предполагает высокую степень автоматизации, в следствие чего понижает денежные расходы. Она показывает эффективный способ навигации в замкнутом пространстве под определенные цели.

* 1. Фовеальное зрение в беспилотных автомобилях

Российская компания Congitive Technologies специализируется на создании систем для автоматизации управления любым автомобилем. В их планах реализовать полный комплекс устройств для автономного вождения уже к 2020 году. Разработку беспилотных автомобилей компания начала в 2014 году, объявив сотрудничество с производителей автомобилей КАМАЗ.[4]

Системы, разработанные Congitive Technologies, не требуют строительства интеллектуальных городов и умных дорог, которые оснащены множеством датчиков – их системы работают в противоположную сторону, пытаясь разобраться в ситуации на дороге так, как это делают люди. В разрабатываемых системах этой компании используется множество различных датчиков и камеры в том числе. Система использует оборудование таким образом, чтобы воспроизвести взгляд водителя, который использует свои глаза, анализируя информацию и ориентируясь на соответствующие данные. Помимо системы слежения за полосой, рассмотренной ранее, для этой цели в автомобилях используется специальный тип компьютерного зрения – фовеальное зрение. Прототипы проходят испытания в России на дорогах без разметки, с ямами, приближая условия тестирования к боевым с целью подготовить систему для работы в трудных условиях и на плохих дорогах по всему миру.

Фовеальным зрением называется особенность человеческого восприятия, при котором четкое видение объектов возможно только в рамках довольно узкого угла зрения. Зона фовеального зрения охватывает пространство, расположенное в центре и немного по сторонам от зрительной оси глаза. Расходящийся угол такого зрения составляет в среднем всего один-два процента. Объекты, находящиеся за пределами фовеального восприятия видятся нечеткими и размытыми. Они располагаются в зонах периферического зрения.

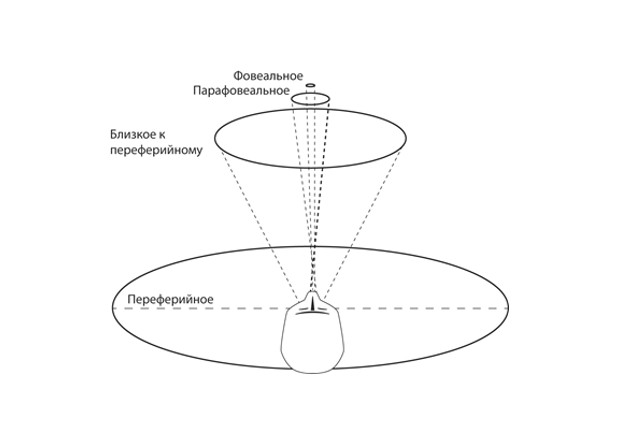


Рисунок 1.4 – Принцип фовеального зрения

Компьютерная технология фовеального зрения работает по похожему принципу. Специальное программное обеспечение выстраивает зону интереса. Из таких зон формируется «виртуальный тоннель», последовательно удаляющиеся участки изображения. Эта зона представляет собой небольшой участок изображения, который при движении подвергается тщательному анализу. При этом в высоком разрешении анализируются лишь объекты и особенности дороги, необходимые для движения: дорожное полотно, расположение обочины, дорожных знаков и съездов. Для построения «виртуального тоннеля» используется принцип внутреннего самоподобия, то есть выявления общих признаков, присущих любому типу дороги.[5]

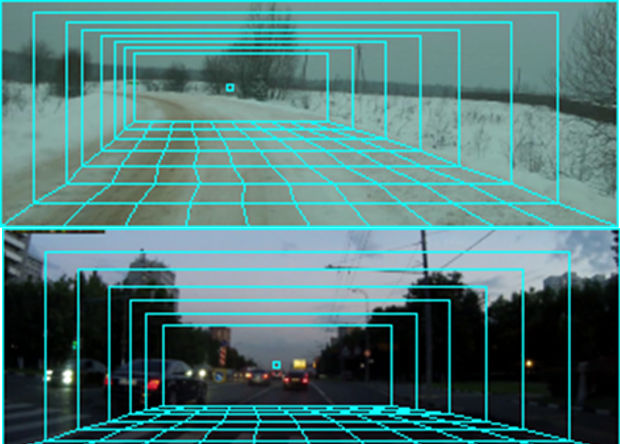


Рисунок 1.5 – Фовеальное зрение автомобиля

Такой способ обработки изображения позволяет не зависеть от линий разметки или других отличительных признаков на дороге при навигации. Это делает разработку компании устойчивой к любым погодным условиям, состояниям дорог, ситуациям, возникающим при движении.

* 1. Навигация робота с использованием обработки изображения и диаграмм Вороного

Этот метод определяет кратчайший путь для робота прямиком к своей цели, избегая препятствий по пути. Окружение захватывается в качестве изображения при помощи веб-камеры. Затем применяются методы обработки изображений для выявления существующих препятствий на пути к конечной точке. С помощью диаграммы Вороного определяются места с препятствиями.

Есть множество точек на плоскости. Выделяется область на плоскости из точек такая, что для всех ближайшей точкой из множества является одна и та же точка , которая называется ячейкой Вороного для точки . Разбиение плоскости на такие ячейки для всех точек называется диаграммой Вороного для множества .

В первую очередь алгоритм выделяет изображение с веб-камеры и выполняет ряд действий по обработке изображения:

1. Применение необходимого уровня яркости
2. Преобразование в градации серого
3. Применение оператора Собеля для нахождения граней
4. Удаление шумов
5. Заливка обнаруженных объектов
6. Вычитание цвета

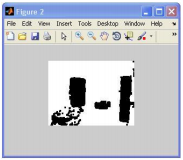
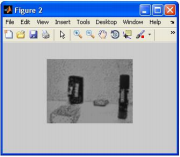


Рисунок 1.5 – Применение методов обработки изображения

Для того, чтобы роботу выбрать кратчайший путь, необходимо выполнить выявить все пути проезда, и ликвидировать невозможных. Рис. 4 иллюстрирует проблему поиска, где указаны 3 вероятных пути

* 1 путь невозможен (пересекается с препятствием)
* 2 путь возможен и является кратчайшим
* 3 путь возможен, но не самый короткий

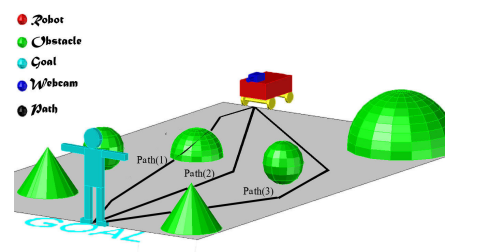


Рисунок 1.6

Очевидно, что количество объектов и количество пути будет различаться, в зависимости от ситуации.

На основе полученных изображений изучаются препятствия и затем строится диаграмма Вороного, где центром каждого сектора является какой-либо обнаруженный роботом объект, затем анализируется кратчайший возможный путь по границам секторов исходя из размеров и расстояния до конечной точки. На рисунке 1.7 показан пример такого пути.

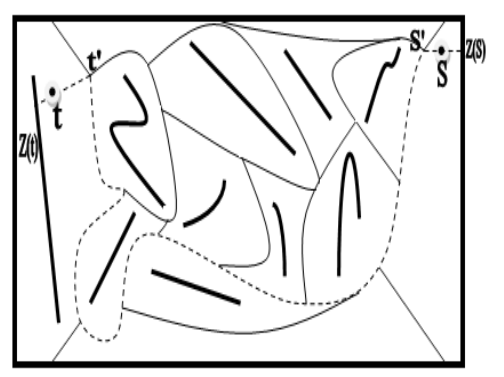


Рисунок 1.7

Такой способ довольно эффективен, но требует полного анализа окружающей обстановки и сложен по вычислениям.

* 1. Навигация на основе сравнения изображений

Этот довольно простой метод использует единственную камеру, прикрепленную к передней части автономного средства. Он заключается в том, что робот сравнивает изображения с уже имеющимися в его базе. Исходя из этого следует, что робота необходимо провести один раз по нужному маршруту вручную, чтобы «научить» и записать изображение в базу. Без всякой калибровки, такой робот способен следовать по пути, делая только сравнения систем координат, вычисленных на этапе обучения с теми, которые вычисляются во время очередной поездки.

Основная идея проста. На рисунке 1.8 робот перемещается из положения в положение вдоль прямой линии, координата ориентира проецируется на плоскость изображения движется монотонно от . Поэтому, пока робот остается на нужном пути, величина координаты также монотонно уменьшается, а ориентир остается на той же стороне плоскости изображения.

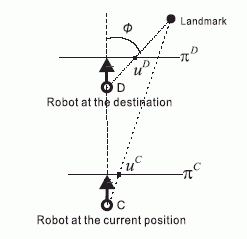


Рисунок 1.8

Это наблюдение приводит нас к простому алгоритму управления: робот непрерывно перемещается вперед, поворачивая направо каждый раз, когда неравенство нарушается и налево, когда нарушается равенство знаков .

На рисунке 1.9 слева располагается изображение в реальном времени, а справа, изображение, хранящееся в базе. Ориентиры показаны красными точками. Как можно заметить на левом изображении некоторые ориентиры имеют окантовку со стрелками, которые показывают необходимость поворота робота в нужную сторону, чтобы совместить ориентиры.

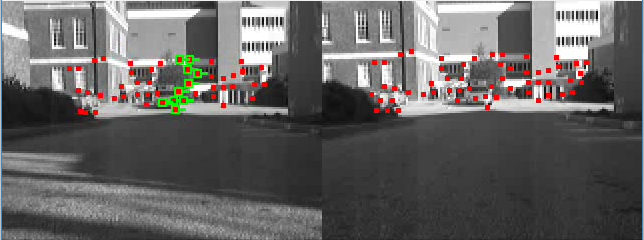


Рисунок 1.9

Этот способ является одним из самых простых, но ограничивается всего-лишь одним путём проезда, что делает его не универсальным, а также чувствительностью к переменам окружающей обстановки робота (тени, погодные условия, появления людей).

1.6 Роль нейронных сетей в компьютерном зрении

В области компьютерного зрения наблюдается постоянный прогресс в течение последних 5 лет. Одной из известных и мощных технологий является сверточная нейронная сеть (СНС). На сегодняшний день, обработке СНС подвергается очень много областей применения компьютерного зрения, например: распознавание жестов, автоматическое распознавание номерных знаков, поиск друзей на фотографиях в социальных сетях, охранные системы и, безусловно, беспилотные автомобили.

Обнаружение объектов считается наиболее основным применением компьютерного зрения. Остальные разработки в области компьютерного зрения достигаются путем небольшого усовершенствования поверх этого. В реальной жизни, каждый раз, когда мы открываем наши глаза, мы можем не задумываясь опознавать объекты. Из-за того, что это очень просто для нас, довольно сложно определить ключевые проблемы, при проектировании систем, похожих на наши органы чувств, но можно выделить несколько очевидных затруднений[6]:

1. Угол расположения объекта

При решении первой проблемы необходимо учитывать тот факт, что объект может находиться под разным углом относительно к наблюдателю. Человек сразу может опознать объект и в какую сторону он повернут, но научить этому компьютер будет сложно.

б). Разница в освещении

Когда освещение располагается позади объекта мы видим только его очертания, но все равно в большинстве случае можем опознать объект, компьютеру же в этой ситуации будет также сложно.

в). Скрытые объекты на изображении

Изображения не обязательно должны быть полными. Какой-то объект может попадать лишь частью или быть загороженным каким-либо другим объектом. Человек опознаёт по небольшой части уже весь объект, но не компьютер.

г). Отличие фона

На некоторых изображениях фон может сливаться с тем, что представлено на переднем плане. Устройство человеческого глаза позволяет почти сразу отличить фон от объекта, в то время как для компьютера это почти невыполнимая задача.

Конечно все эти задачи можно решать отдельно и добиться хороших результатов, но необходимо обойти все эти проблемы сразу, одним решением. Здесь на помощь и приходят нейронные сети. СНС работают путем разбиения изображения на небольшие фрагменты и затем обрабатывая, и совмещая их глубже в сети. Сеть состоит из множества слоев, где обрабатывается информация. Первый слой, как правило, обнаруживает очертания и грани объектов, затем последующие слои пытаются объединить найденные края в более простые формы и, в конечном счете, в шаблоны различных положений объекта, освещение, масштабы и т.д. Глубокие СНС способны работать со сложными изображениями и даёт довольно точные результаты их обработки.

Таким образом нейронными сетями можно добиться необходимых результатов в компьютерном зрении. С их помощью изображение становится больше похожим на то, которое видит человеческий глаз, что позволяет эффективно разработать систему ориентирования в пространстве для автономных транспортных средств.

**Глава 2. Используемые методы для реализации**

Для реализации алгоритма навигации внутри помещений с использованием камеры применяется ряд методов обработки изображения, которые будут приведены в данной главе. Эти методы заключаются в преобразованиях изображения, обнаружении и выделении на нём границ объектов.

* 1. Методы обработки изображений

Для обнаружения информации (например, определенных объектов) на изображениях необходимо производить предварительную обработку изображения, которая может производиться несколькими различными способами. Каждый рассмотренный в этой главе приём работает с изображениями в градациях серого.

* + 1. Бинаризация изображения

Для некоторых задач обработки изображений цвет не играет большой роли, поэтому для кодирования пикселя можно оставить только два значения: 0 или 1, что означает, что точка может находиться в двух состояниях, черная или белая соответственно. Бинаризация проводится с помощью простого порогового преобразователя, где проверяется значение пикселя, и если оно перешло порог, значит пиксель делается черным, иначе белым.[7]

* + 1. Морфологические преобразования

Исходное изображение может содержать шум и другие артефакты. Для борьбы с этими проблемами применяют операции математической морфологии. Каждый пиксель(точка) исходного изображения рассматривается с некоторой окрестностью. Размер и её форма задаются какой-либо произвольной структурной фигурой, но в большинстве случаем ей является квадрат 3х3. Обычно применяются две основные операции, сужение и расширение, и их комбинации.[7]

* Сужение (Erode) – логическое И, точка принимает определенное значение если все точки в окрестности имеют такое же значение.
* Расширение (Dilate) – логическое ИЛИ, точка принимает определенное значение, если хотя бы одна точка в окрестности принимает такое же значение.
* Закрытие (closing) – последовательное выполнение расширения и сужения.
* Раскрытие (opening) – последовательное выполнение сужения и расширения.
  + 1. Фильтры

Часто, во избежание тех же шумов, на изображение накладываются различные фильтры. Применение фильтра – значит изменение значения каждой точки в зависимости от её окрестности, а размер этой окрестности называют размером фильтра.

Фильтры строятся на матричных преобразованиях и реализуются через свёрточный фильтр, который имеет следующий вид:

где, *f* – исходное изображение, *g* – ядро фильтра

Проще говоря используется обычное умножение матрицы окрестности изображения на матрицу фильтра.[7]

Примерами фильтров можно рассмотреть Гауссовый фильтр размытия, сдвиг изображения. В Гауссовом размытии матрица фильтра заполняется нормальным распределением по функции Гаусса, а при сдвиге изображения матрица фильтра заполняется единицами в той стороне, куда необходимо произвести сдвиг.

* 1. Методы выделения контуров

2.2.1 Оператор Собеля

Оператор Собеля — дискретный дифференциальный оператор, который вычисляет приближённое значение для функций яркости пикселей изображения. Оператор часто используется для выделения границ объектов на изображении.[8]

Оператор вычисляет градиент яркости изображения в каждой точке. Таким образом находится направление увеличения яркости и то, насколько резко эта яркость изменяется. Если изменение яркости происходит резко, то, следовательно, можно предположить, что эта точка находится на грани. Математически, градиентом функции двух переменных для каждой точки изображения — двумерный вектор, компонентами которого являются производные яркости изображения по горизонтали и вертикали. В каждой точке изображения градиентный вектор ориентирован в направлении наибольшего увеличения яркости, а его длина соответствует величине изменения яркости. Это означает, что результатом оператора Собеля в точке области постоянной яркости будет нулевой вектор, а в точке, лежащей на границе областей различной яркости — вектор, пересекающий границу в направлении увеличения яркости.

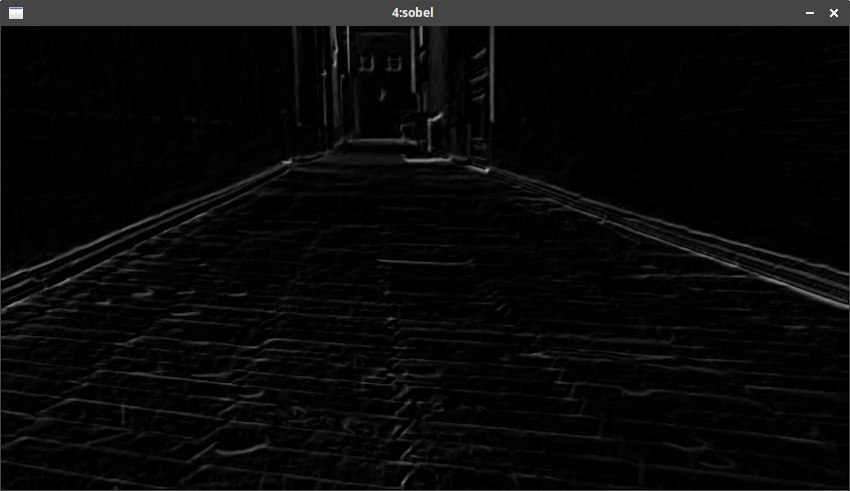


Рисунок 2.1 – Оператор Собеля

* + 1. Оператор Превитт

Оператор Превитт — метод выделения границ в обработке изображений, который вычисляет максимальный отклик на множестве ядер свёртки для нахождения локальной ориентации границы в каждом пикселе.[8] Этот способ был создан доктором Джудит Превитт для обнаружения границ на медицинских изображениях. Из одного ядра можно получить восемь, переставляя вращательно коэффициенты. Каждый результат будет чувствителен к направлению границы от 0° до 315° с шагом в 45°, где 0° соответствует вертикальной границе. Максимальный ответ каждого пиксела есть значение соответствующего пиксела в выходном изображении. Значения его лежат между 1 и 8, в зависимости от номера ядра, давшего наибольший результат.

Этот метод выделения границ также называется подстановкой шаблонов границ (англ. edge template matching), поскольку изображению сопоставляется набор шаблонов, и каждый представляет некоторую ориентацию границы. Величина и ориентация границы в пикселе тогда определяется шаблоном, который лучше всех соответствует локальной окрестности пиксела.



Рисунок 2.2 – Оператор Превитт

* + 1. Оператор Канни

На сегодняшний день этот алгоритм является самым мощным среди способов выделения контуров. Джон Канни изучил математическую проблему получения такого фильтра, который будет оптимален по критериям выделения, локализации и минимизации нескольких откликов одного края. Это означает, что детектор игнорирует ложные границы, но при этом реагирует на верные, точно определяя их линию (без её фрагментирования) и реагирует на каждую границу всего один раз, что позволяет избежать участков совокупности границ. Канни ввел понятие Non-Maximum Suppression (подавление не-максимумов), которое означает, что пикселями границ объявляются точки, в которых достигается локальный максимум градиента в направлении вектора градиента.

Алгоритм Канни состоит из нескольких шагов:

* + - 1. Сглаживание. Чаще всего применяется Гауссово размытие для удаления шумов.
      2. Поиск градиентов. Там, где градиент становится максимальным значением, помечаются границы.
      3. Подавление не-максимумов. Отбираются только локальные максимумы.
      4. Двойная пороговая фильтрация. Пороговая функция определяет потенциальные границы.
      5. Трассировка области неоднозначности. Итоговые границы определяются путём подавления тех краёв, которые не связаны с сильно определенными границами.

Таким образом при обработке изображения данным алгоритмом можно получить двоичное изображение с тонкими границами, где ни одна из них не будет повторяться.[9]

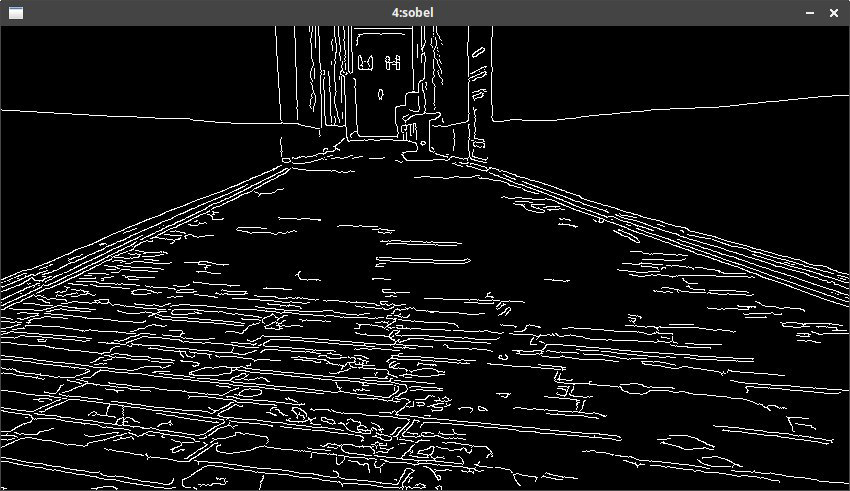


Рисунок 2.3 – Оператор Канни

Подводя итог, можно выделить плюсы и минусы работы каждого из алгоритмов обнаружения граней, которые приведены в таблице 2.1

Таблица 2.1 – Сравнение алгоритмов выделения граней

|  |  |  |
| --- | --- | --- |
| Алгоритм | Плюсы | Минусы |
| Оператор Собеля | * Мало затрат * Рассматриваются только соседние точки | * Нечеткие и темные границы * Много гладких линий * Медленная скорость работы |
| Оператор Превитт | * Мало затрат | * Много гладких линий * Медленная скорость работы |
| Оператор Канни | * Границы получаются чёткими, единой линией * Имеется быстрая оптимизированная функция OpenCV | * Наиболее затратный |

Сравнивая результаты работы каждого из алгоритмов поиска контуров на изображении можно сделать вывод, что самым эффективным из всех является оператор Канни. Стоит помнить, что все алгоритмы чувствительны к «зашумленности» изображения, поэтому перед выявлением границ необходимо воспользоваться общими методами обработки изображений, рассмотренными ранее.

* 1. Преобразование Хафа для обнаружения линий на изображении

Преобразование было изобретено Ричардом Дуда и Питером Хартом в 1972 году, и было названо "обобщенное преобразование Хафа", на основании соответствующего патента 1962 года Пола Хафа. Алгоритм стал популярен в компьютерном сообществе после его публикации Х. Баллардом в статье журнала 1981 года под названием «обобщенное преобразование Хафа для обнаружения произвольных фигур».

Преобразование Хафа – это метод, который используется для того, чтобы обнаружить формы конкретной фигуры на изображении. Объекты часто находятся в какой-либо стандартной геометрической форме, поэтому классическое преобразование Хафа наиболее часто используется для обнаружения различных кривых, таких как линии, окружности, эллипсы и т. д. В результате преобразование Хафа может быть использовано в ситуациях, где простое аналитическое описание объектов не представляется возможным. Несмотря на свои ограничения домена, классического преобразования Хафа (далее без классической префикс) сохраняет многие приложения, как большинство изготавливаемых деталей (и многие анатомические части исследованных медицинских изображений) содержат границ объектов, которые могут быть описаны с помощью регулярных кривых. Основное преимущество преобразования Хафа заключается в том, что оно устойчиво к разрывам в границах объектов и практически не поддаётся воздействию шумов на изображении.

В алгоритме преобразования Хафа используется аккумуляторный массив, который обладает такой размерностью, сколько неизвестных параметров в уравнении какого-либо семейства кривых. Например, при обнаружении прямых линий, которые описываются формулой 2.1

где – координаты, а – коэффициенты

Следовательно, для каждой прямой необходимо вычислить значения параметров a и b. При этом в этом аккумуляторном массиве в элементах M[A,B] накапливаются значения, которые указывают на вероятность наличия прямой, где A и B соответствуют дискретным значениям a и b.[10]

Этот массив используется в алгоритме Хафа для проверки каждого пикселя и его окрестности. Сначала определяется находится ли данный пиксель на выраженном крае. Если находится там, то вычисляются параметры искомой кривой, проходящей через данный пиксель. После оценки параметров прямой в данном пикселе они дискретизируются для получения соответствующих значений A и B, и значение массива M[A,B] увеличивается. В некоторых реализациях увеличение выполняется на единицу, в других на величину мощности края в обработанном пикселе. После обработки всех пикселов выполняется поиск локальных максимумов в аккумуляторном массиве. Точки локальных максимумов соответствуют параметрам наиболее вероятных прямых на изображении.

Имеющийся массив позволяет определить параметры бесконечно протяжённых линий, но с его помощью невозможно определить, где именно начинаются и заканчиваются отрезки этих линий. Для их определения можно завести еще один массив, которые содержит точки, внесшие вклад в получение массива M[A,B]. По содержанию нового массива можно найти присутствующие на изображении отрезки или сегменты кривых.

Но для программной реализации важно знать еще несколько деталей об этом алгоритме. Формула 2.1 не подходит для представления вертикальных прямых. Гораздо удобнее представлять прямые в полярных координатах заданных формулой 2.2

где – это длина перпендикуляра к прямой, а – угол между перпендикуляром и горизонтальной осью.

В системе координат изображения оси направлены вдоль строк и столбцов изображения.

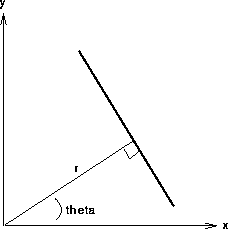


Рисунок 2.1 – Представление прямой в полярной системе координат

Рассмотрим пример, где необходимо определить, находятся ли все 3 точки на одной прямой.

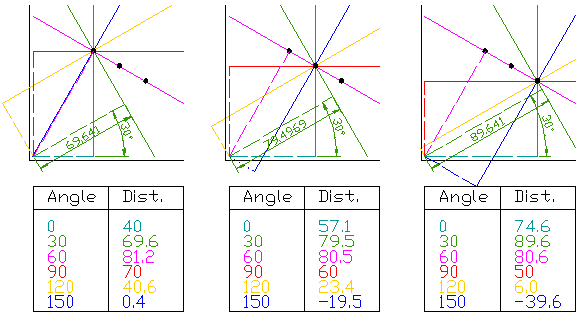


Рисунок 2.2 – Пример с 3 точками на одной прямой

На данном изображении через каждую точку проведено по шесть прямых, с разным углом. К каждой из прямых построен перпендикуляр из начала координат. Полученные углы и расстояние заносятся в таблицу, которая является результатом преобразования Хафа, то есть аккумуляторным массивом, описанным ранее. Эти таблицы могут служить данным для графического представления в пространстве Хафа. Построим график зависимости длины перпендикуляра от угла по таблицам.

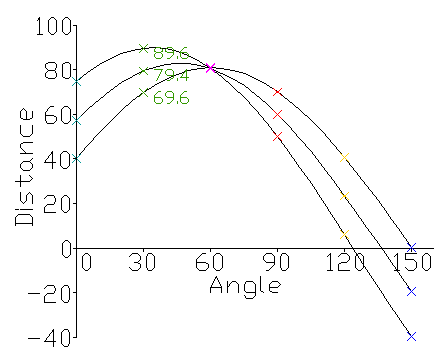


Рисунок 2.3 – График зависимости длины перпендикуляра от угла

На данном графике можно заметить, что синусоиды, построенные для каждой точки с прошлого изображения, пересекаются в одном месте. Эта точка пересечения является одной из 6 прямых, построенных ранее, следовательно, все три точки лежат на этой прямой.

В данной главе были рассмотрены методы используемые для обработки изображения, выделения границ объектов, нахождения линий. Используя всех их вместе можно добиться хорошего и правильного результата в обработке каждого кадра с исходного видеофайла.

**Глава 3. Программная реализация**

3.1 Постановка задачи

В данной работе была поставлена задача изучения возможности навигации робота внутри помещения при помощи методов обработки изображения. Для этого требуется разработать программный модуль который в последствии должен применяться на автономных транспортных средствах. Следовательно, система должна обрабатывать видеофайл, полученный в реальном времени с камеры установленной на роботе, затем отдавать ему данные для передвижения в нужную сторону.

Не имея такого робота, для реализации данной задачи была написана программа для тестирования возможности навигации внутри помещений, которая обрабатывает предварительно записанный видеофайл. В данной главе описаны: алгоритм работы программы, использованные средства разработки, такие как необходимые библиотеки, среда разработки, язык программирования, функции и результаты выполнения программы.

3.1 Используемые средства разработки

3.1.1 Среда разработки Code::Blocks (IDE)

Среда разработки Code::Blocks является свободно распространяемой, кросс-платформенной интегрированной средой разработки с открытым исходным кодом, которая поддерживает множество компиляторов, включая GCC, clang и Visual с++. Она разработан на языке C++ с использованием wxwidgets, в качестве GUI-инструментария. Используя архитектуру плагинов, открываются множество возможностей этой среды. В настоящее время, Code::Blocks ориентирован на языки программирования С, C++ и Fortran.[12]

История этой среды разработки начинается с релиза версии 1.0rc1 от 25 июля 2005 года, и 1.0rc2 25 октября 2005 года, но вместо того, чтобы сделать окончательный релиз, разработчики проекта начали добавлять множество новых функций, вследствие чего финальный релиз неоднократно откладывался. Первый стабильный релиз состоялся 28 февраля 2008 года, номер версии изменился до 8.02. Схема версионности была позаимствована с оболочки для ОС Linux – Ubuntu, где основной и дополнительный номера обозначают год и месяц выпуска. На данный момент последней стабильной версией является выпуск 16.01, однако для наиболее актуальной версии пользователь может загрузить относительно стабильную «ночную» сборку или же скачать исходный код из SVN. Также компания Jennic Limited распространяет версию Code::Blocks предназначенную для работы с микроконтроллерами.

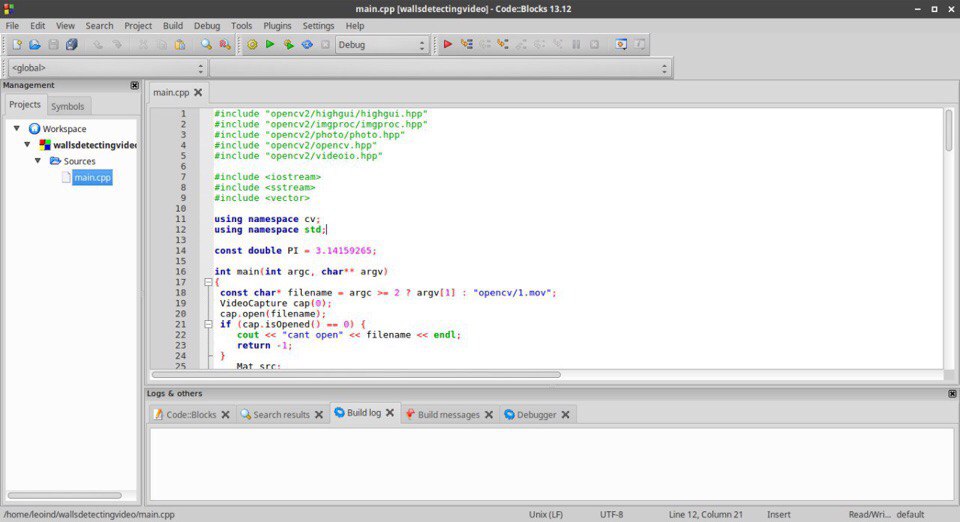


Рисунок 3.1 – Окно IDE Code::Blocks

3.1.2 Библиотека OpenCV

Opencv – это свободно распространяющаяся библиотека, направленная на разработку приложений реального времени, которые используют компьютерное зрение. Она имеет реализацию на языках программирования C++, С, Python и Java, а также поддерживает многие популярные операционные системы, такие как Windows, Linux, Mac OS, iOS и Android. Она написана преимущественно на языках С/С++ в оптимизированном виде. Также библиотека может пользоваться преимуществами многоядерной обработки при помощи поддержки технологии opencl. Известная во всем мире, OpenCV насчитывает более 47 тысяч человек из числа пользователей, а предполагаемое количество загрузок превышает 14 миллионов.[12]

Библиотека насчитывает более 2500 оптимизированных алгоритмов, которые используются для применения в классическом компьютерном зрении и машинном обучении. Эти алгоритмы могут быть использованы для обнаружения и распознавания лиц, определения объектов, классификации действий и движений человека на видео, отслеживании движущихся объектов, извлечении 3D-моделей объектов, поиска 3D точек со стереокамер, поиска похожих образов из баз данных, удаления эффекта красных глаз с изображений, снятых с использованием вспышки, следования за движениями глаз, распознаванию пейзажей и установление их связей с дополненной реальность, а также многое другое.[13]

История проекта берет начало с официального запуска в 1999 году. Изначально OpenCV была инициирована компанией Intel для поддержки ресурсоемких приложений, в том числе и в режиме реального времени, захватывающих области компьютерного зрения. Основной вклад в проект вложили эксперты по оптимизации и сотрудники по производительности компании Intel в России.

3.1.3 Язык программирования C++

C++ — это язык программирования общего назначения. Он имеет императивные, объектно-ориентированные и универсальные возможности для программирования, а также предоставляет средства для низкоуровневых манипуляций с памятью. Он был разработан с уклоном в сторону системного программирования, для больших, встраиваемых, и систем с ограниченными ресурсами. С точки зрения производительности, эффективности и гибкости использования, этот язык практически идеален. Многие другие языки программирования были созданы под влиянием C++, включая такие известные, как C#, D, Java и новые версии С.[14]

История этого языка программирования начинается с 1979 года, когда Бьярне Страуструп, датский компьютерный специалист, начал работать над языком с названием «Си с классами», предшественником C++. Страуструп обнаружил, что язык программирования Simula имеет особенности, которые были очень полезны для крупных разработчиков программного обеспечения, но язык был слишком медленным для практического использования, в то время как, например, язык BCPL являлся очень быстрым языком, но слишком низкого уровня, чтобы быть тогда пригодным для крупного развития программного обеспечения.

3.2 Алгоритм разработанной программы

Для успешного решения задачи был разработан алгоритм обработки, включающий необходимые этапы обработки изображения, основываясь на методах, описанных в Главе 2. Для решения задачи необходимо обработать кадр с видео таким образом, чтобы после обработки можно было автоматически принять решение о направлении направить автономного транспортного средства в нужную сторону. Блок схема алгоритма представлена на рисунке 3.2.

Первым шагом необходимо выделить кадр из видео. Это можно сделать как из потокового видео в реальном времени, так и из заготовленного предварительно видеофайла (для задач тестирования алгоритма и программы).

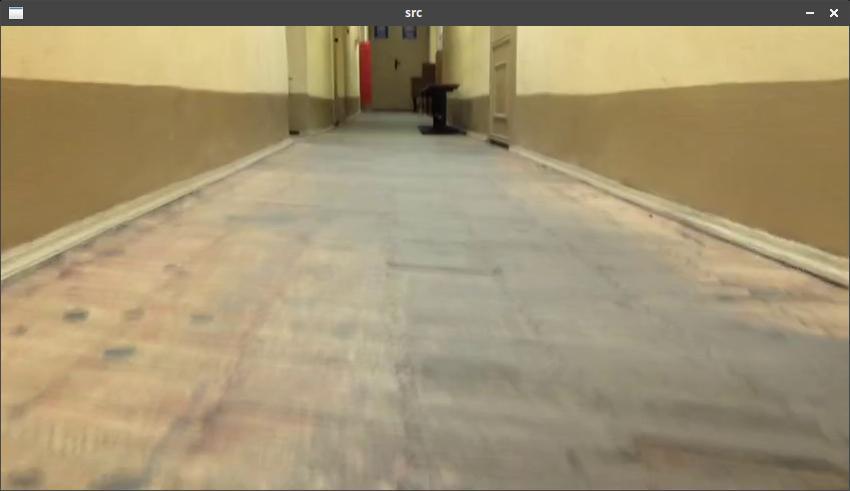
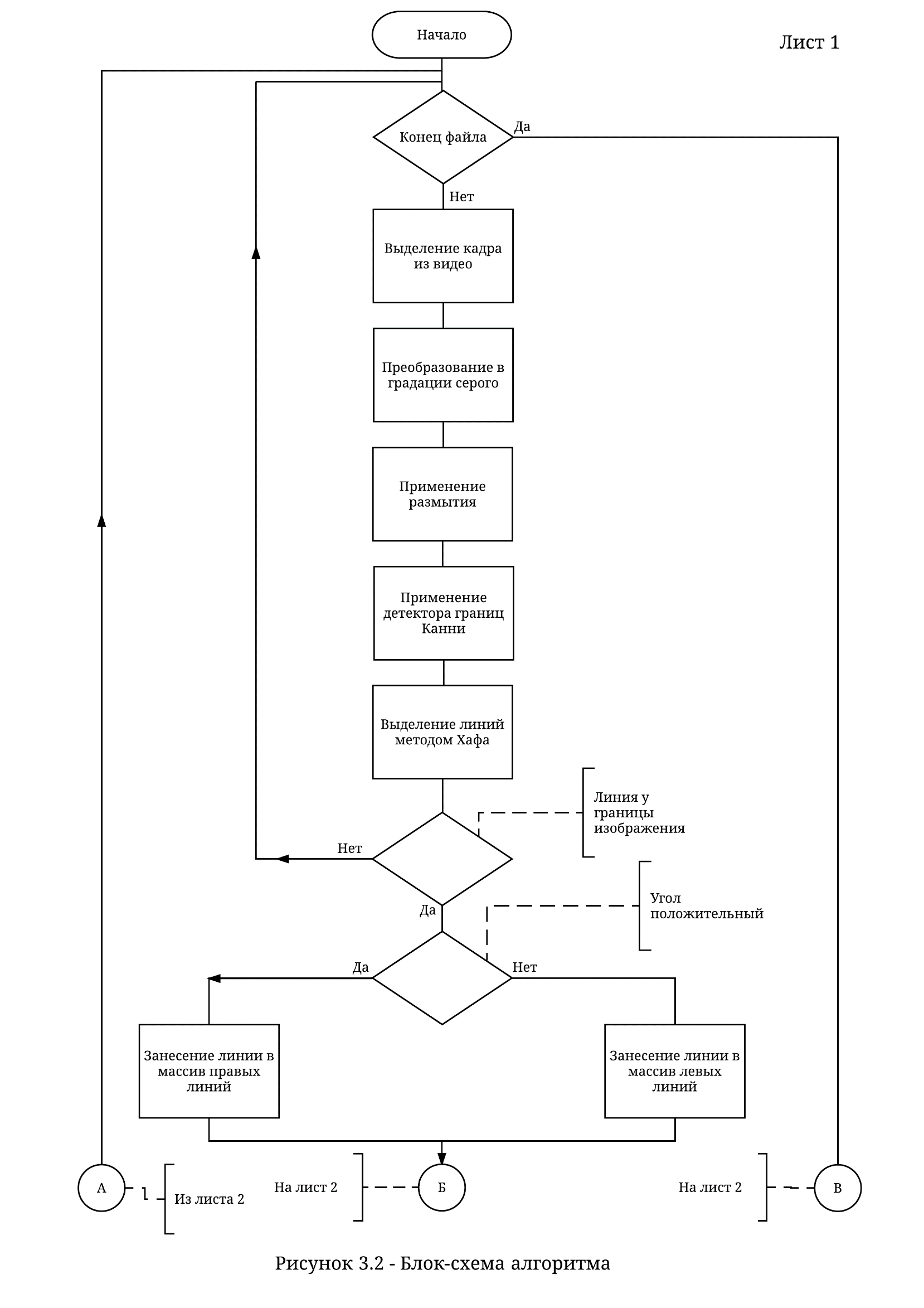


Рисунок 3.3 – Исходное изображение



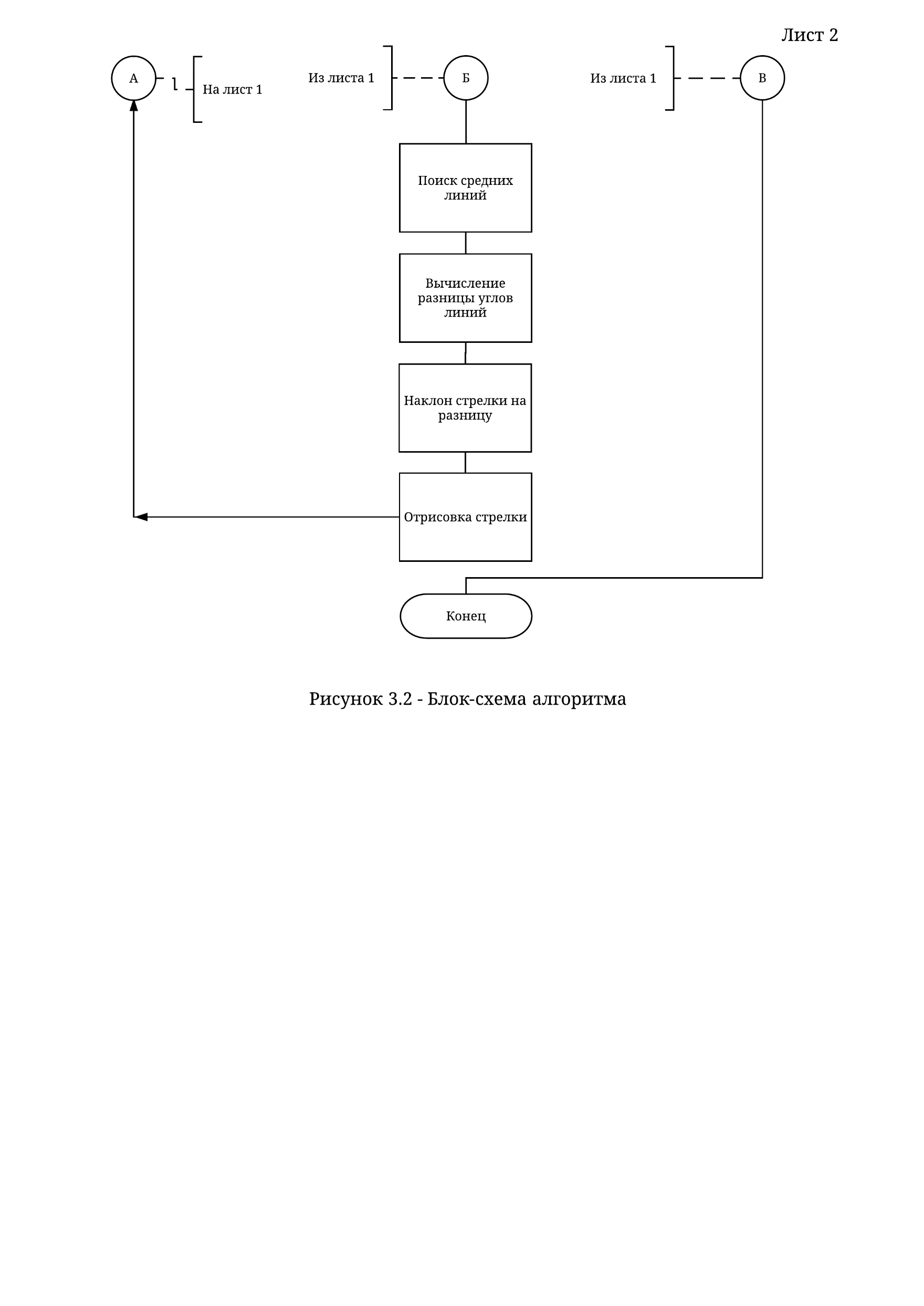




Рисунок 3.4 – Применение градаций серого

Затем применяется размытие для сглаживания границ объектов, которое показано на рисунке 3.5.



Рисунок 3.5 – Применение размытия

Затем применяется оператор Канни, предназначенный для выделения границ, который показан на рисунке 3.6. Размытие на предыдущем шаге применяется для того, чтобы он сработал более точно.

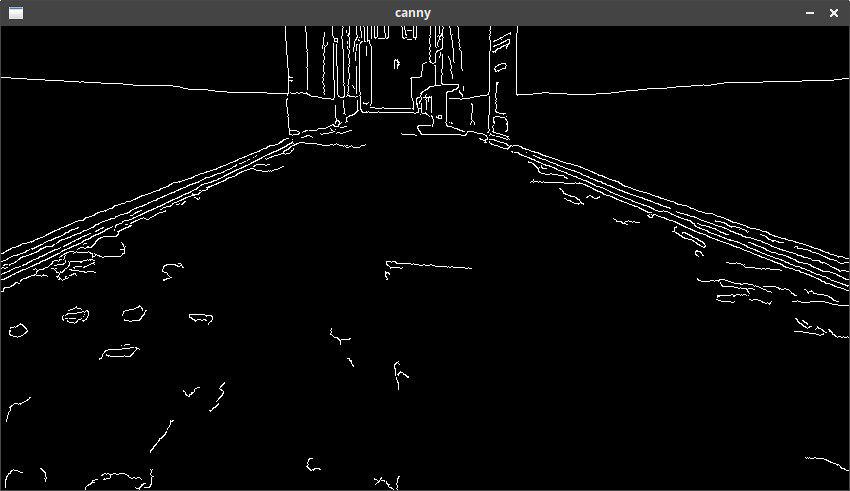


Рисунок 3.6 – Применение оператора Канни

Выделив границы объектов и переведя таким образом изображение в бинарный цвет, можно применить преобразование Хафа (рисунок 3.7) для поиска всех линий на изображении, которые собираются в массив.

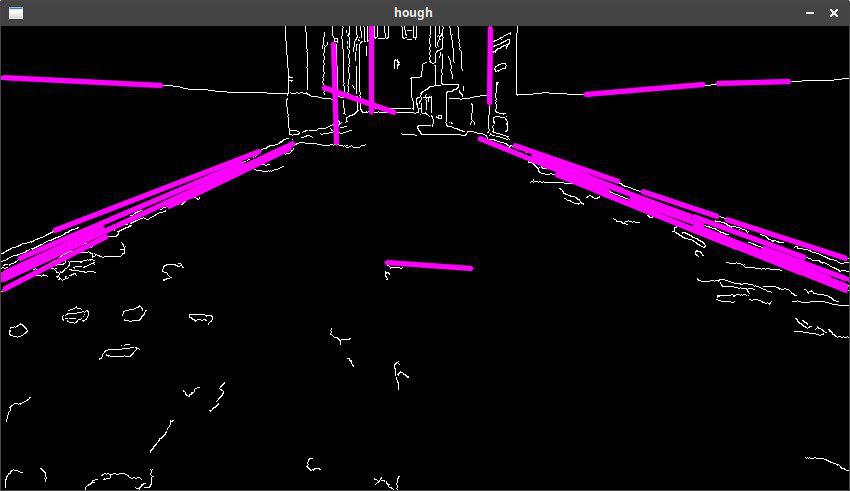


Рисунок 3.7 – Выделение всех линий методом Хафа

Так как задача не предусматривает использование всех линий их необходимо отфильтровать. Учитывается положение и наклон линий относительно горизонтали, чтобы отделить предполагаемые стены, границы в которых движется робот. Результат фильтрации показан на рисунке 3.8

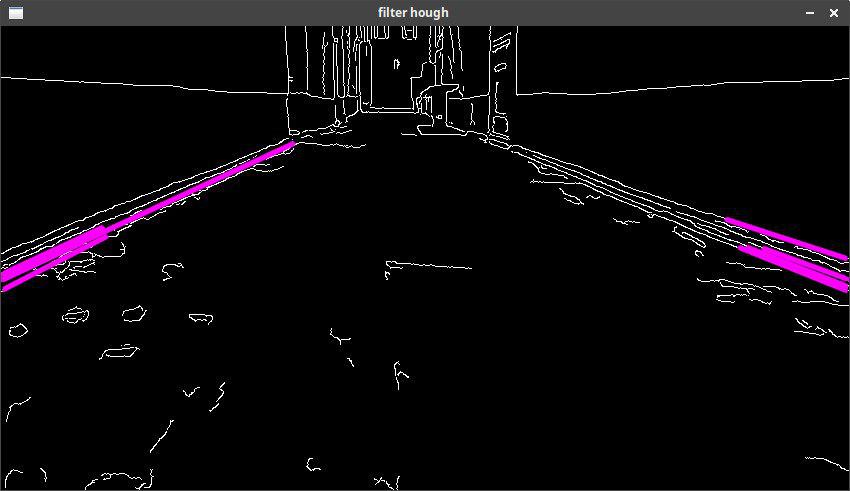


Рисунок 3.8 – Фильтрация линий

Выделив только необходимые линии необходимо вычислить к какой стороне относится линия и вычислить самую близкую к середине. Затем между двумя линиями считается разница наклона относительно горизонтали и мнимый привод колёс, что является стрелкой на результирующем видео, поворачивается на вычисленный угол в необходимую сторону.

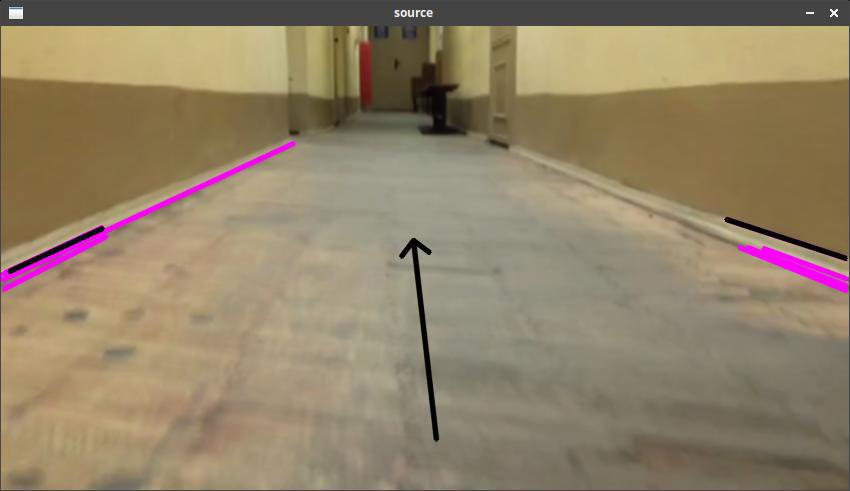
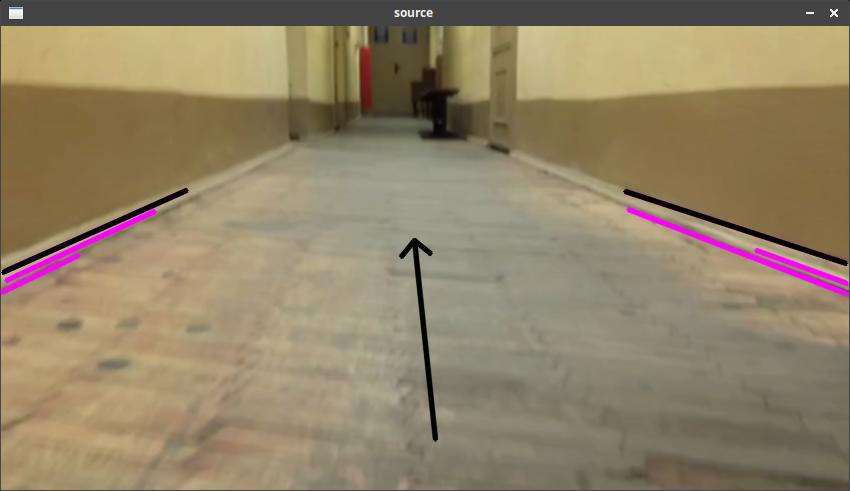
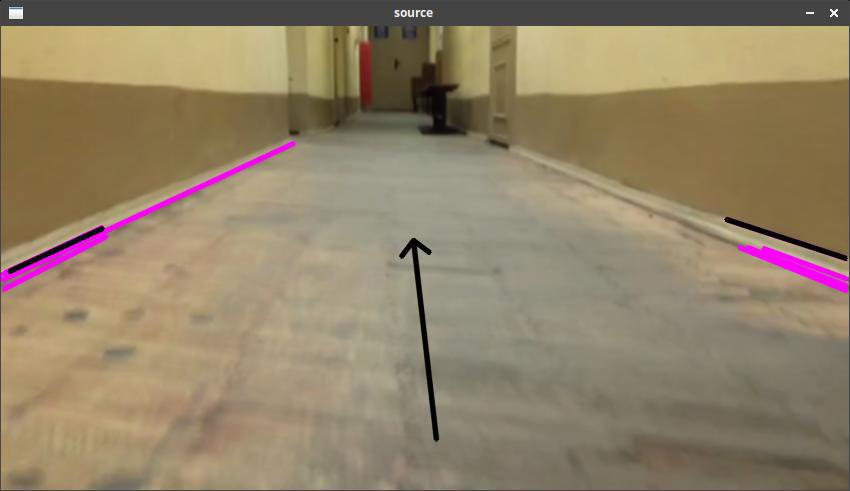


Рисунок 3.9 – Конечный результат

Далее алгоритм повторяется до конца видеофайла или пока не будет остановлен. На рисунке 3.10 представлена последовательность кадров результирующего видео.



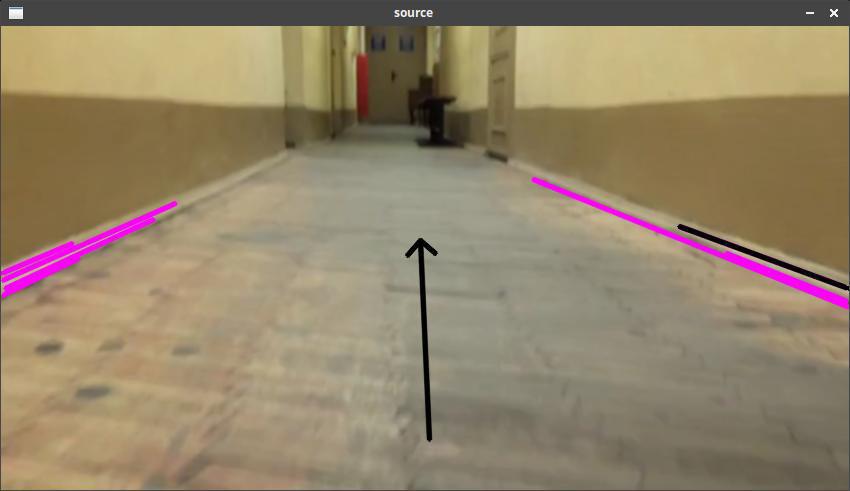
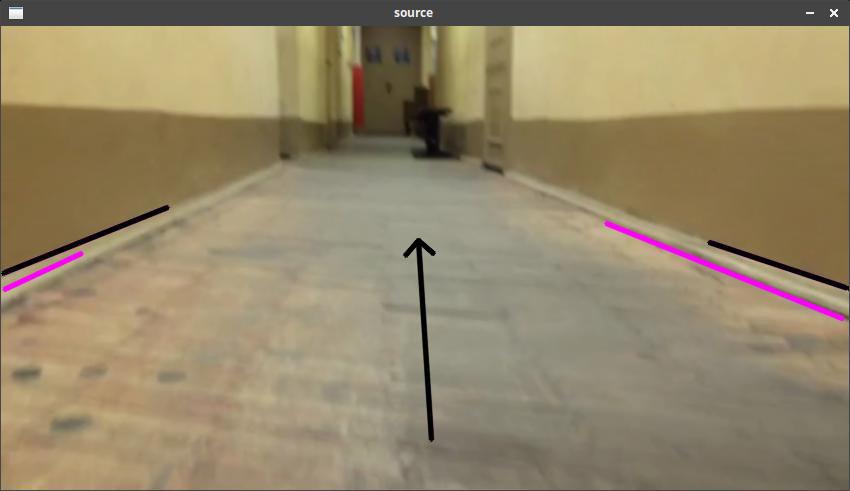


Рисунок 3.10 – Последовательность кадров выходного видео

3.3 Программная реализация

Программа для выполнения задачи построена на основе алгоритма, описанного выше. Она написана на языке C++ с использованием библиотеки OpenCV, имеет единственный исходный файл и выполнена в виде консольного приложения, но с возможностью вывода результирующих изображений на экран в привычном для пользователя интерфейсе.

Ниже будут описаны функции и структуры данных библиотеки OpenCV, которые используются в программе.

3.5.1 Структура данных Mat

Основной структурой данных в OpenCV является Mat. Она представляет собой двумерную матрицу, работа с которой производится с помощью различных функций. Так как OpenCV работает в основном с изображениями, эта структура данных может представлять матрицу пикселей изображения. Для изображений в градациях серого элемент матрицы представлен 8-битным беззнаковым числом. Для цветного изображения в формате RGB таких чисел 3, по одному на каждую компоненту цвета.[15]

3.5.2 Функция VideoCapture::open

Эта функция позволяет открыть видеофайл или подключить захватывающее устройство. Она принимает единственный параметр filename.[16]

* filename – Имя видеофайла для открытия (например, video.avi) или последовательности изображений (например, img\_%02d.jpg, что соответствует файлам img\_00.jpg, img\_01.jpg, img\_02.jpg и т.д.)

Пример:

VideoCapture cap(0);

cap.open(filename);

if (cap.isOpened() == 0) {

cout << "cant open" << filename << endl;

return -1;

}

3.5.3 Функция Canny

Эта функция позволяет создать изображение с обнаруженными гранями используя оператор Канни.[17]

Пример:

Mat res, dst;

Canny(res, dst, 20, 50, 3);

* res —исходное изображение для обработки
* dst — одноканальное изображение для хранения границ, найденных функцией
* threshold1 — порог минимума, в нашем случае 20
* threshold2 — порог максимума, в нашем случае 50
* aperture\_size — размер для оператора Собеля, стандартное значение 3

3.5.4 Функция HoughLinesP

Эта функция позволяет выделить линии на изображении вероятностным методом Хафа, который от классического метода(HoughLines) отличается тем, что проводится анализ лишь части точек исходного изображения, что позволяет работать более эффективно в случае изображений с несколькими длинными линейными сегментами.[17]

Пример:

Mat dst;

vector<Vec4i> lines;

HoughLinesP(dst, lines, 1, CV\_PI/180, 50, 70, 10 );

* dst — изображение для обработки
* lines — массив для сохранения найденных линий
* rho — разрешение по дистанции, в нашем случае 1
* theta — разрешение по углу (в радианах), в нашем случае 1 градус
* threshold — пороговый параметр, минимальное количество «пересечений» синусоид, в нашем случае 50
* minLinLength — минимальная длина линии, в нашем случае 70
* maxLineGap — максимальный промежуток между сегментами линии, чтобы считать их одним целым, в нашем случае 10

3.5.5 Функция line

Для визуализации найденных линий алгоритмом Хафа необходимо использовать функцию line, которая строит заданную линию по 4 координатам.[18]

Пример:

Mat dst, cdst1;

vector<Vec4i> lines;

HoughLinesP(dst, lines, 1, CV\_PI/180, 50, 70, 10 );

for( size\_t i = 0; i < lines.size(); i++ )

{

Vec4i l = lines[i];

line( cdst1, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(255,0,255),3, CV\_AA);

}

* cdst1 — изображение на котором рисуются линии
* pt1 — первая точка линии, задается через структуру Point
* pt2 — вторая точка линии, задается через структуру Point
* color — цвет линии, в нашем случае фиолетовый (R = 255, G = 0, B = 255)
* thickness — толщина линии, в нашем случае 3
* lineType — тип линии, в нашем случае используется сглаженная линия (CV\_AA - antialiased)

3.5.6 Функция вычисления угла наклона

Для определения границ, в которых должен двигаться робот необходимо фильтровать линии по углу наклона, для этого была реализована функция вычисления этого угла, и фильтрация отношения линий к правой или левой стороне изображения. Он вычисляется по формуле 3.1

где координаты точек на прямых.

Функция angles:

void angles (vector<vec4i> lines) {

for( size\_t i = 0; i < lines.size(); i++ )

{

Vec4i l = lines[i];

double angle = atan2(l[3]-l[1], l[2]-l[0]) \* 180.0 / CV\_PI;

if ((angle < -5 && angle > -45 && l[0] < 15)) {

left.push\_back(l);

//cout << "right" << endl;

itr++;

}

if (angle > 5 && angle < 45 && l[2] > (dst.size().width - 15)) {

right.push\_back(l);

//cout << "left" << endl;

itl++;

}

}

}

3.6 Результаты работы программы

Для запуска программы необходима установленная библиотека OpenCV. Для работы программы требуется исходный видеофайл к которому необходимо указать путь. Так как программа выполнена в виде консольного приложения при её запуске открывается окно, которое представлено на рисунке 3.11, с просьбой указать путь.

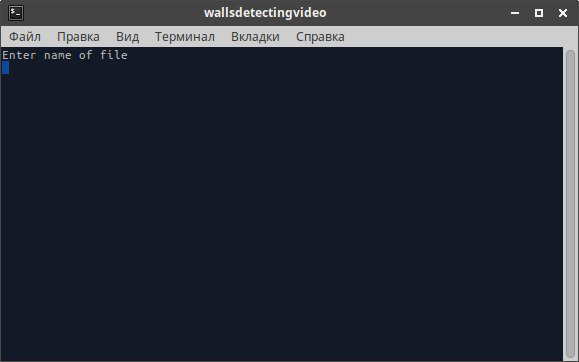


Рисунок 3.11 – Окно при запуске приложения

Далее пользователю предоставляется возможность обработать видео целиком или же какое-то количество кадров. При обработке полного видео в папке opencv появляется файл output.mov с результатом обработки

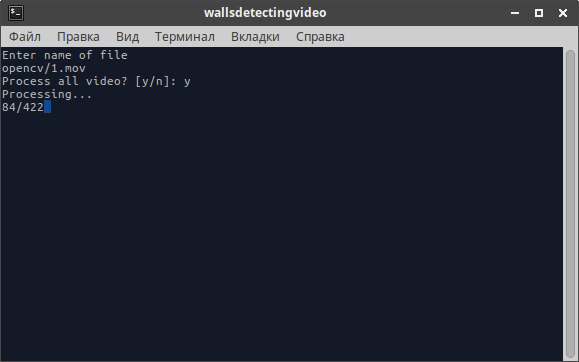


Рисунок 3.12 – Подтверждение об обработке видео полностью

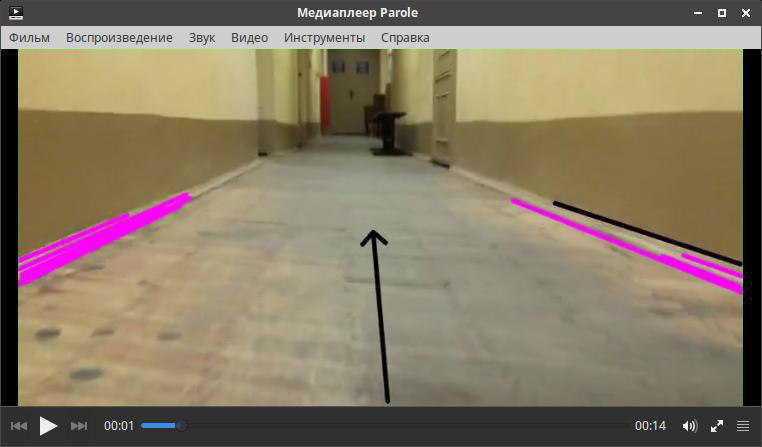


Рисунок 3.13 – Выходной видеофайл

Если ограничить количество кадров для обработки, пользователю необходимо ввести нужное ограничение, а также предоставляется возможность вывода шагов обработки последнего кадра. Диалог с пользователем представлен на рисунке 3.14

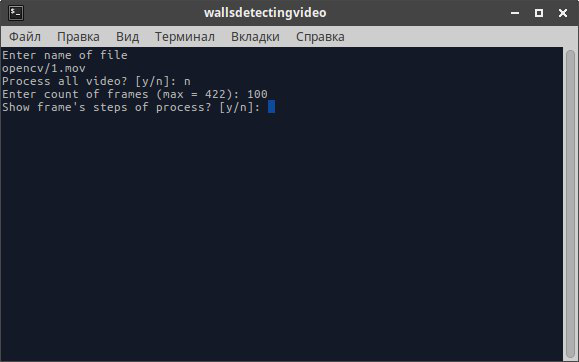


Рисунок 3.14 – Подтверждение о выводе этапов обработки изображения

При выборе данной возможности в конце работы программы по нажатию любой кнопки на клавиатуре появляются окна с этапами обработки изображения.

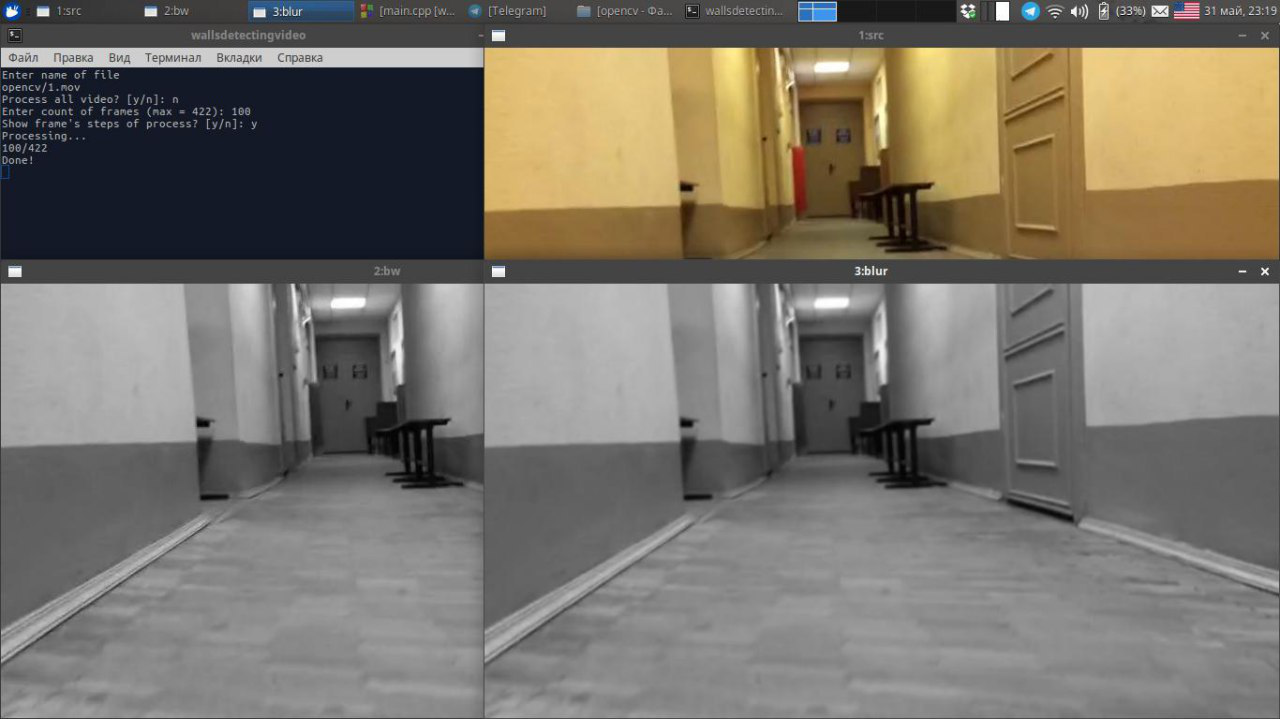


Рисунок 3.15 – Вывод этапов обработки изображения

В конце работы программы также записывается файл output.mov с ограниченным пользователем количеством кадров.

Такая визуализация даёт представление о работе программы, этапах обработки изображения и возможности её применения на том или ином автономном устройстве.

3.7 Дальнейшая разработка и применение программы

Данная программа обрабатывает каждый кадр исходного видеофайла, что занимает довольно много времени. Для её применения на автономного робота необходимо понизить количество обрабатываемых кадров. Опытным путём было выяснено, что для успешного использования количество обрабатываемых кадров в секунду необходимо сократить до 10.

В том числе необходимо скорректировать оптимальную скорость передвижения робота, так как изображение с камеры при очень быстрой скорости может смазываться, а робот не будет успевать реагировать на быструю смену обстановки.

Также следует учитывать, что привод колес не имеет возможности быстро их поворачивать, следовательно, необходимо сглаживать движение «стрелки» в зависимости от возможности привода. Это позволит избегать ситуаций, показанных на рисунке 3.16

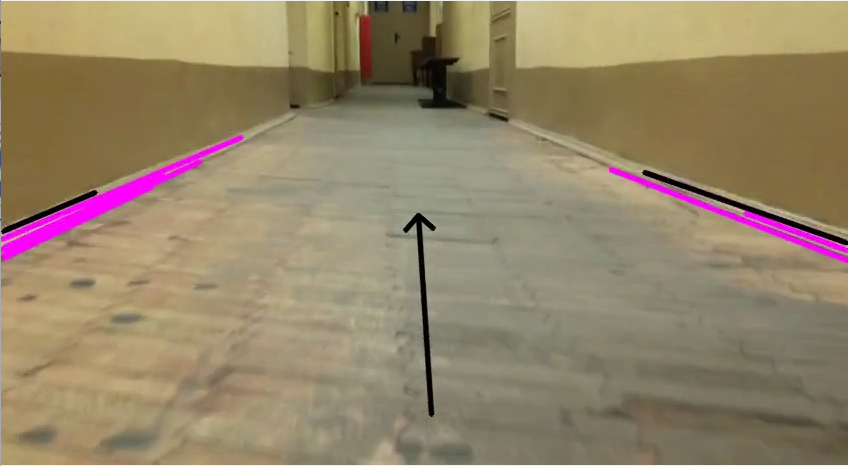
 

Рисунок 3.16 – Проблема «сглаживания»

При тестах программы на разных видеофайлах можно заметить, что результат работы алгоритма может зависеть от качества, освещения и других факторов, влияющих на изображение, поэтому при дальнейшей разработке имеется возможность оптимизации алгоритма под различные условия съемки.

В алгоритме в том числе можно предусмотреть механизм калибровки параметров программы под определенные условия окружающей среды. Эти параметры могут зависеть от освещения, возможностей камеры, цвета объектов и других ситуаций. Калибровку можно проводить анализируя цветовые пространства изображения в зависимости от данных которых можно производить изменения параметров алгоритма.

Также имеется возможность использования программы совместно с другими средствами навигации, что позволяет более эффективно использовать алгоритм и улучшить способность робота ориентироваться в пространстве.

**Заключение.**

В данной работе было проведено исследование возможности навигации автономного транспортного средства в пространстве при помощи камеры. Были рассмотрены имеющиеся технологии, позволяющие роботам ориентироваться в пространстве на основе которых были оценены возможности для реализации алгоритма.

В результате был выведен алгоритм на основе методов обработки изображения, по которому было разработано консольное приложение для тестирования возможности навигации робота в пространстве.

При разработке были протестированы различные ситуации, в которых может оказаться робот, что привело к результатам, которые дают понять, что алгоритм может быть использован для использования на реальных автономных транспортных средствах.

Разработанное приложение может быть использовано как в целях прямого назначения – тестирования «глаз» робота, так и в целях изучения методов и этапов обработки изображения в компьютерном зрении.

**Список использованных источников**

1. <https://www.ijcsi.org/papers/IJCSI-8-6-1-402-408.pdf>
2. <https://globaljournals.org/GJCST_Volume12/1-Automated-Road-Lane-Detection.pdf>
3. <http://www.mwpvl.com/html/kiva_systems.html>
4. <https://vc.ru/n/cognitive-cpilot>
5. http://cognitivepilot.com/
6. https://www.analyticsvidhya.com/blog/2016/04/deep-learning-computer-vision-introduction-convolution-neural-networks/
7. <https://www.lektorium.tv/course/22847>
8. <https://habrahabr.ru/post/114452/>
9. <http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html>
10. <http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm>
11. <http://www.codeblocks.org/>
12. <http://opencv.org/>
13. <http://docs.opencv.org/>
14. Бьерн Страуструп. Язык программирования С++
15. <http://docs.opencv.org/2.4/modules/core/doc/basic_structures.html>
16. <http://docs.opencv.org/3.1.0/d8/dfe/classcv_1_1VideoCapture.html>
17. <http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html>
18. http://docs.opencv.org/2.4/modules/core/doc/drawing\_functions.html