

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»
(НГТУ)

Институт радиоэлектроники и информационных технологий
Направление подготовки (специальность) 09.03.01 Информатика и вычислительная техника
Направленность (профиль) образовательной программы Вычислительные машины,
комплексы, системы, сети
Кафедра Вычислительные системы и технологии


ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

бакалавра

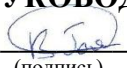
Студента Черкас Наталии Александровны группы 16-B-1 на тему:

«Программная система распознавания разметки и знаков дорожного движения для
мобильного робота»

СТУДЕНТ:

 Черкас Н.А.
(подпись) (фамилия, и., о.)
03.07.2020
(дата)

РУКОВОДИТЕЛЬ:

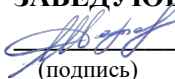
 Гай В.Е.
(подпись) (фамилия, и., о.)
03.07.2020
(дата)

РЕЦЕНЗЕНТ:

(подпись) (фамилия, и., о.)

(дата)

ЗАВЕДУЮЩИЙ КАФЕДРОЙ:

 Жевнерчук Д.В.
(подпись) (фамилия, и., о.)
03.07.2020
(дата)

КОНСУЛЬТАНТЫ:

1. По _____

(подпись) (фамилия, и., о.)

(дата)

2. По _____

(подпись) (фамилия, и., о.)

(дата)

3. По _____

(подпись) (фамилия, и., о.)

(дата)

ВКР защищена _____
(дата)

протокол № _____
с оценкой _____

**МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»
(НГТУ)**

Кафедра Вычислительные системы и технологии



УТВЕРЖДАЮ
Зав. кафедрой
Жевнерчук Д.В.
«12» мая 2020 г.

**ЗАДАНИЕ
на выполнение выпускной квалификационной работы**

по направлению подготовки (специальности) 09.03.01 Информатика и вычислительная техника

студенту Черкас Наталии Александровне группы 16-B-1
(Ф.И.О.)

1. Тема ВКР Программная система распознавания разметки и знаков дорожного движения для мобильного робота

(утверждена приказом по вузу от _____ № _____)

2. Срок сдачи студентом законченной работы 3 июля 2020 г

3. Исходные данные к работе Данные из видеопотока; набор данных для обучения и тестирования классификатора

4. Содержание расчетно-пояснительной записки (перечень вопросов, подлежащих разработке)

Введение

1. Техническое задание

2. Анализ технического задания

3. Разработка структуры системы поиска знаков дорожного движения, светофоров и дорожной разметки

4. Разработка программных средств

5. Тестирование системы

Заключение

Список литературы

5. Перечень графического материала (с точным указанием обязательных чертежей)

1. Общая схема работы системы

2. Схема алгоритма распознавания знаков дорожного движения и светофоров

3. Схема алгоритма распознавания дорожной разметки




4. Примеры изображений из набора данных для обучения


5. Изображения работы разработанной системы

6. Консультанты по ВКР (с указанием относящихся к ним разделов)

Нормоконтроль Гай В.Е.

7. Дата выдачи задания 4 апреля 2020 г.

Код и содержание Компетенции	Задание	Проектируемый результат	Отметка о выполнении
ПК-1, Способность разрабатывать модели компонентов информационных систем, включая модели баз данных и модели интерфейсов «человек – электронно-вычислительная машина»	Разработать структурную схему системы	Разработана структурная схема системы	Выполнено 
ПК-2, Способность разрабатывать компоненты аппаратно-программных комплексов и баз данных, используя современные инструментальные средства и технологии программирования	Разработать алгоритмы работы системы, реализовать их на одном из языков программирования с помощью современной среды разработки	Алгоритмы работы системы разработаны на языке программирования Python в среде разработки PyCharm	Выполнено 
ПК-3, Способность обосновывать принимаемые проектные решения, осуществлять постановку и выполнять эксперименты по проверке их корректности и эффективности	Осуществить отбор лучших результатов, оценить их качество по нескольким критериям	Проведено тестирование работы системы на нескольких тестовых изображениях и оценено качество полученных результатов	Выполнено 

Руководитель  Гай В.Е.
(подпись)

Задание принял к исполнению 04.04.2020
(дата)

Студент  Черкас Н.А.
(подпись)

**МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»
(НГТУ)**

АННОТАЦИЯ

к выпускной квалификационной работе

по направлению подготовки 09.03.01 Информатика и вычислительная техника
(код и наименование)

студента Черкас Наталии Александровны группы 16-B-1
(Ф.И.О.)

по теме Программная система распознавания разметки и знаков дорожного движения для мобильного робота

Выпускная квалификационная работа выполнена на 55 страницах, содержит 41 рисунок, библиографический список из 6 источников, 2 приложения.

Актуальность: На сегодняшний день очень актуальной темой является разработка беспилотных автомобилей, так как их использование может понизить количество автомобильных катастроф, повысить безопасность вождения, а также увеличить производительность поездок. Такие автомобили могут, например, уведомлять водителя о запрещающих знаках дорожного движения, о красном сигнале светофора, либо о схождении автомобиля с полосы движения

Объект исследования: Видеопоток, изображения

Предмет исследования: Алгоритмы распознавания объектов и алгоритмы обработки входных данных

Цель исследования: Разработка программной системы распознавания знаков дорожного движения, светофоров и дорожной разметки

Задачи исследования: Анализ существующих методов реализации системы: выбор средств для разработки системы; разработка системы и проверка ее на работоспособность

Методы исследования: Поиск, обработка и анализ научно-технической информации из различных источников

Структура работы: Введение, пять разделов, заключение, список литературы и два приложения

Во введении цель и актуальность работы

В 1 разделе «Техническое задание» описаны технические требования к разрабатываемой системе

Во 2 разделе «Анализ технического задания» производится выбор программных средств для реализации системы; обзор существующих подходов к решению поставленной цели и выбор наилучшего

В 3 разделе «Разработка структуры системы» осуществляется разработка структурной схемы, алгоритмов работы программы; описываются все этапы работы алгоритмов

В 4 разделе «Разработка программных средств» разрабатываются программные средства для решения поставленной цели

В 5 разделе «Тестирование системы» приводится описание тестовых изображений, методик тестирования и результатов вычислительного эксперимента


В заключении Приводятся основные выводы по работе

Выводы:

1. Разработана программная система распознавания знаков дорожного движения, светофоров и дорожной разметки
2. Тестирование системы подтвердило ее работоспособность

Рекомендации:





1. Дальнейшее развитие проекта
3. Оптимизация существующего кода

 / Черкас Н.А.
подпись студента /расшифровка подписи

« 3 » июля 2020 г.

Оглавление

Введение.....	5
1 Техническое задание.....	6
1.1 Назначение разработки и область применения.....	6
1.2 Технические требования.....	7
2 Анализ технического задания.....	8
2.1 Выбор операционной системы.....	8
2.2 Выбор языка программирования.....	9
2.3 Выбор среды разработки.....	10
2.4 Обзор существующих методов решения задач распознавания светофоров, знаков дорожного движения и дорожной разметки.....	12
3 Разработка структуры системы поиска знаков дорожного движения, светофоров и дорожной разметки.....	15
3.1 Общая структура разрабатываемой системы.....	15
3.1.1 Алгоритм распознавания знаков дорожного движения и светофоров...	16
3.1.2 Алгоритм распознавания дорожной разметки.....	17
3.2 Создание набора данных для обучения каскада Хаара.....	18
3.2.1 Положительные примеры.....	18
3.2.2 Отрицательные примеры.....	19
3.3 Каскад Хаара.....	21
3.3.1 Признаки Хаара.....	21
3.3.2 Интегральное представление изображений.....	23
3.3.3 Сканирующее окно.....	24
3.3.4 Алгоритм Adaboost.....	24
3.4 Описание алгоритма распознавания дорожной разметки.....	26
3.4.1 Получение и предобработка изображения.....	26

					ВКР-НГТУ-09.03.01-(16-В-1)-017-2020 (ПЗ)			
Изм.	Лист	№ докум.	Подпись	Дата	Программная система распознавания разметки и знаков дорожного движения для мобильного робота Пояснительная записка	Лит.	Лист	Листов
Разраб.		Черкас Н.А.						
Провер.		Гай В.Е.		03.07.2020			3	69
Н. контр.		Гай В.Е.		03.07.2020		НГТУ кафедра ВСТ		
Утверд.		Жевнерчук Д.В.		03.07.2020				

3.4.2 Поиск дорожной разметки	28
3.4.3 Принятие решения и отображение результата.....	31
4 Разработка программных средств	34
4.1 Программная реализация алгоритма распознавания светофоров и знаков дорожного движения.....	34
4.2 Программная реализация алгоритма распознавания дорожной разметки	40
5 Тестирование системы.....	43
5.1 Тестирование системы распознавания светофоров и знаков дорожного движения	43
5.2 Тестирование системы распознавания дорожной разметки.....	48
Заключение	51
Список литературы	52
Приложение 1	53
Приложение 2	64

Введение

В настоящее время происходит автоматизация практических всех видов деятельности в большинстве случаев для того, чтобы упростить жизнь людям.

На сегодняшний день очень актуальной темой является разработка беспилотных автомобилей. Это автомобили, оборудованные системой автоматического управления и способные передвигаться без участия человека.

Беспилотные автомобили нужны, прежде всего для того, чтобы понизить количество автомобильных катастроф, в которых ежегодно гибнут десятки тысяч людей. Автомобили с возможностью автономного движения помогают повысить безопасность вождения, увеличить производительность поездок, снизить стресс водителей при управлении транспортным средством. Такие автомобили могут оказать помощь водителям, чтобы избежать аварий или к примеру, оценить оптимальный расход топлива. Именно поэтому, многие крупные компании, такие как Tesla, Apple, Google, Nvidia и Waymo, инвестировали большие суммы денег в разработку аппаратного и программного обеспечения, необходимого для создания автономных транспортных средств [5].

Транспортные средства с возможностью полностью автономного вождения обязательно должны уметь решать такие задачи как: распознавание знаков дорожного движения, светофоров и дорожной разметки.

Данная работа посвящена разработке надежного детектора знаков дорожного движения, светофора и дорожной разметки.

					ВКР-НГТУ-09.03.01-(16-В-1)-017-2020 (ПЗ)	Лист
						5
Изм	Лист	№ докум.	Подп.	Дата		

1 Техническое задание

1.1 Назначение разработки и область применения

Разрабатываемая программная система предназначена для распознавания знаков дорожного движения, светофора и дорожной разметки в реальном времени или на видеозаписи.

Область применения разрабатываемой системы:

Разрабатываемая система может использоваться для помощи водителю, уведомляя его, например, о запрещающих знаках дорожного движения, о красном сигнале светофора, либо о схождении автомобиля с полосы движения.

					ВКР-НГТУ-09.03.01-(16-В-1)-017-2020 (ПЗ)	Лист
						6
Изм	Лист	№ докум.	Подп.	Дата		

1.2 Технические требования

Рассмотрим требования, предъявляемые разрабатываемой системой к ЭВМ:

1. Операционная система с графическим интерфейсом;
2. Требования к аппаратному обеспечению определяются операционной системой;
3. Дисплей;
4. Мышь;
5. Видеокамера (необязательно).

Функционал, которым должна обладать разрабатываемая система:

1. В систему должна быть загружена видеозапись, либо подключена видеокамера;
2. Система должна обработать исходные данные и на их основе подготовить дополнительные данные, необходимые для дальнейшей работы;
3. С помощью набора признаков система должна определить знаки дорожного движения, дорожную разметку и светофор;
4. Система должна показать распознанные объекты в реальном времени, если подключена камера или на видеозаписи;
5. Система должна сохранить результирующий видеофайл, на котором будут отображены все найденные объекты.

					ВКР-НГТУ-09.03.01-(16-В-1)-017-2020 (ПЗ)	Лист
						7
Изм	Лист	№ докум.	Подп.	Дата		

2 Анализ технического задания

2.1 Выбор операционной системы

Первое, на что нужно обратить внимание при планировании программного проекта – это выбор операционной системы. Для того чтобы достичь наилучшей производительности разрабатываемой системы, необходимо правильно подобрать операционную систему. Также немаловажно, чтобы работа в выбранной операционной системе была удобна для разработчика. Самыми популярными операционными системами на сегодняшний день являются Windows, Linux и Mac OS. Рассмотрим каждую из перечисленных операционных систем и решим, какая из них больше всего подойдет для решения поставленной задачи.

Windows – это платная операционная система, которая была разработана компанией Microsoft. Данная операционная система имеет простой и удобный графический интерфейс. Windows является очень популярной операционной системой, вследствие чего под Windows разрабатывается огромное количество программ и работает данная система практически со всеми устройствами.

Linux – это семейство Unix-подобных операционных систем на базе ядра Linux. Данная система является бесплатной, а ее ключевая особенность – открытый исходный код, благодаря чему есть возможность изменять и настраивать систему под свои нужды. Существует очень много разновидностей операционных систем на базе Linux, самые распространённые из них: Linux Mint, Ubuntu, Debian, Fedora, CentOS и другие. Из недостатков данной операционной системы можно отметить, что не все популярные и привычные для обычных пользователей программы есть в Linux.

Mac OS – это платная операционная система производства Apple. Данная операционная система является второй по популярности после Windows. Mac OS – система с закрытым исходным кодом. Установить данную систему легально возможно только на компьютер от производителя Apple. Для данной системы существует достаточно большое количество программ. Mac OS отлично подходит под веб-разработку. Недостаток данной операционной системы – это высокая стоимость компьютеров Apple, на которых данная операционная система установлена [5].

Для решения задачи распознавания знаков дорожного движения, светофора и дорожной разметки отлично подойдет операционная система Windows, так как она удобна в использовании и отвечает ранее заявленным требованиям.

2.2 Выбор языка программирования

Следующим важным этапом в планировании программного проекта является выбор языка программирования. Существует большое количество различных языков программирования, которые имеют разные свойства, свои плюсы и минусы. В зависимости от поставленной задачи необходимо правильно выбрать язык программирования, для того чтобы в будущем при разработке программной системы не столкнулся с большими сложностями. Рассмотрим самые популярные языки программирования.

Python – это высокоуровневый универсальный язык программирования, который подходит для многих платформ и позволяет решать самые разнообразные задачи. Разработка на данном языке займет гораздо меньше времени, чем на многих других языках, так как придется писать меньше кода. Python имеет минимум служебных символов, динамическую типизацию, автоматическое управление памятью и максимально понятный синтаксис. Но несмотря на свою простоту, данный язык является одним из мощнейших. Стоит отметить, что Python очень популярен в решении задач машинного обучения. Данный язык так же имеет большое сообщество и огромное число модулей, написанных разработчиками со всего мира.

C++ - высокоуровневый компилируемый, статически типизированный язык программирования общего назначения. Данный язык имеет большие возможности при работе с памятью и поэтому часто применяется в системном программировании – при создании операционных систем, драйверов, антивирусов и так далее. C++ отлично подходит для программ, где важна высокая скорость работы и производительность. Данный язык имеет сложный синтаксис и объемную спецификацию языка, поэтому код, написанный на языке C++ будет иметь большое количество строк. Исходный код C++ компилируется в машинный код и поэтому исполняемые файлы данного языка программирования не зависят от платформы.

Java – высокоуровневый строго-типизированный объектно-ориентированный язык программирования. Этот язык является основным в разработке программ для Android. Синтаксис Java основан на языке C++, но является менее сложным и благодаря этому код, написанный на языке Java, можно использовать эффективнее для достижения конкретных результатов. В отличие от C++, Java независим от платформы и в конкретных задачах это имеет огромное значение. В отличие от Python и C++, Java потребляет большее количество памяти. На данный момент Java имеет высокую скорость работы, большую по сравнению с Python, но меньшую по сравнению с C++ [5, 6].

Для разработки программной системы был выбран язык программирования Python, так как разработка на данном языке займет меньше времени, а код, написанный на Python будет выглядеть более лаконично, чем на Java и C++. Также этот язык программирования имеет высокую скорость, достаточную в условиях данной задачи.

					ВКР-НГТУ-09.03.01-(16-В-1)-017-2020 (ПЗ)	Лист
						9
Изм.	Лист	№ докум.	Подп.	Дата		

2.3 Выбор среды разработки

Большинство программ, предназначенных для разработки программного обеспечения, поддерживают большое количество разных языков программирования и имеют множество функций. Поэтому такие программы могут занимать много места в памяти, отнимать много времени для установки или требовать хороших знаний для правильного использования. Рассмотрим самые известные среды разработки и выберем наилучшую из них для выбранного языка программирования.

PyCharm – это интегрированная среда разработки, которая предназначена именно для языка программирования Python. Данная среда разработки имеет бесплатную и платную версии. В PyCharm очень легко создавать новые проекты и открывать уже имеющиеся. Данная среда поддерживает разработку на Python напрямую – нужно просто открыть новый файл и сразу начинать писать код. Запуск и отладка кода может происходить прямо в PyCharm. Стоит отметить, что данная среда разработки имеет очень приятный и удобный интерфейс. А также PyCharm быстро запускается и не тормозит при открытии больших файлов, в отличие от некоторых аналогичных программ.

Sublime Text – это популярная среда разработки, доступная на всех платформах и поддерживающая язык программирования Python. Данный текстовый редактор отличается легкостью в использовании и высокой скоростью работы. В Sublime Text присутствует большой набор пакетов, расширяющих возможности синтаксиса и редактирования. Однако установить дополнительные Python-пакеты может быть достаточно сложно. К недостатку данного редактора можно отнести невозможность отладки и запуска кода. Также Sublime Text не является бесплатной программой, хотя есть возможность использовать пробный период.

Eclipse + PyDev. Eclipse – это среда разработки, предназначенная для программирования на языке Java. Но существует большое количество расширений, делающих данную среду разработки полезной для разных задач. Одним из таких расширений является PyDev, которое представляет собой интерактивную консоль Python. Для полноценного использования данной среды разработки потребуется опыт, так как интерфейс Eclipse достаточно сложный.

Visual Studio – среда разработки от компании Microsoft, доступная на платформах Windows и Mac OS. Существует две версии Visual Studio: бесплатная и платная. У данной среды разработки существует собственный набор расширений, в том числе Python Tools for Visual Studio. Это Python-расширение, позволяющее писать на языке программирования Python и включающее в себя отладку, автодополнение и другие инструменты. Если планируется писать программу только на языке Python, то данная среда разработки – не лучший выбор.

					ВКР-НГТУ-09.03.01-(16-В-1)-017-2020 (ПЗ)	Лист
						10
Изм.	Лист	№ докум.	Подп.	Дата		

Spyder – среда разработки для программирования на языке Python, оптимизированная для data science. Spyder – это инструмент, подходящий скорее для какой-либо конкретной цели, нежели для постоянной работы, так как данная среда разработки не является достаточно функциональной. Программа Spyder является бесплатной и доступна на всех платформах.

Jupyter Notebook – мощный инструмент для интерактивной разработки и представления проектов в области data science. Данный инструмент сохраняет вместе код, изображения, комментарии и графики и в случае работы с большими данными — это очень удобно. У Jupyter Notebook есть поддержка большого количества языков программирования, но чаще всего используется именно Python [5, 6].

PyCharm был выбран в качестве среды разработки, так как данная среда разработана именно для языка Python и нет необходимости устанавливать дополнительные расширения. Также данная среда обладает отличным функционалом, достаточным в условиях поставленной задачи, а также высокой скоростью работы.

					ВКР-НГТУ-09.03.01-(16-В-1)-017-2020 (ПЗ)	Лист
						11
Изм	Лист	№ докум.	Подп.	Дата		

2.4 Обзор существующих методов решения задач распознавания светофоров, знаков дорожного движения и дорожной разметки

Существует два основных метода распознавания объектов на изображении:

1. Поиск объекта на изображении исходя из особенностей его цвета и формы;
2. Поиск объекта на изображении с помощью алгоритмов машинного обучения.

Рассмотрим подробнее каждый из этих подходов к решению поставленной задачи.

Алгоритм распознавания объектов с помощью первого метода следующий:

1. Первый этап - предварительная обработка изображения. Сюда входит преобразование изображения из пространства RGB в другое цветовое пространство, например, HSV или HSI. Это нужно для того, чтобы уменьшить различные помехи, вызванные изменением освещенности, воздействием погодных условий и так далее;
2. Следующий этап - применение к изображению различных масок для поиска определенного цвета на изображении. К примеру светофор состоит из красного, желтого и зеленого цветов, следовательно, для каждого цвета необходимо создать граничные значения и сформировать маски, фильтрующие заданный цветовой диапазон;
3. Для того чтобы уменьшить количество найденных регионов, которые не относятся к искомому объекту, часто применяются морфологические операции – наращивание, позволяющее увеличить контур вокруг найденного региона и эрозия, позволяющая контур уменьшить;
4. Далее необходимо провести контурный анализ, то есть найти границы всех ранее найденных регионов. Существует большое количество разных методов по нахождению контуров, например, детектор границ Канни, оператор Собеля и другие;
5. Следующим этапом является фильтрация контуров по площади для улучшения результата распознавания;
6. Заключительный этап – это перебор всех контуров и поиск шаблона, который максимально похож на данный контур. То есть найденный объект сравнивается с заранее созданными шаблонами по некоторой выбранной метрике. Наиболее подходящим шаблоном будет тот, который имеет меньше всего точек, отличающихся от исследуемого объекта. Распространенным шаблонным методом является попиксельное сравнение объекта и шаблона, так как данный способ имеет высокую точность и очень прост в реализации.

Несмотря на различную предварительную обработку изображений, данный метод в условиях плохой освещенности или нехорошей погоды будет работать плохо или не работать вовсе. Или наоборот, при солнечной погоде на объекте может появиться блик, перекрывающий часть изображения, вследствие чего эта часть будет восприниматься камерой не как искомый цвет, который входит в заданный цветовой диапазон. Из-за этого объект может быть не найден [6].

Машинное обучение – это поиск каких-либо закономерностей по набору эмпирических данных. То есть на основе большого количества данных, полученных с какого-то конкретного объекта, выявляются взаимосвязи, которые присутствуют в данном объекте.

Задача классификации – это разновидность машинного обучения, суть которой состоит в том, чтобы из множества объектов, которые разделены на разные классы по какому-либо признаку, определить для произвольного объекта принадлежность к одному из этих классов [5].

Рассмотрим некоторые подходы для решения задач распознавания с помощью машинного обучения:

1. Каскад Хаара

Это один из методов поиска объектов на изображении, основанный на машинном обучении. Обученный каскад Хаара принимает на вход изображение и определяет присутствует ли на нем искомый объект. То есть данный метод делит входные данные на два класса:

- 1) на изображении есть объект;
- 2) на изображении объекта нет.

Распознавание объектов происходит с помощью признаков Хаара. Признак Хаара состоит из смежных прямоугольных областей, которые располагаются на изображении, далее суммируются интенсивности пикселей в двух областях, затем вычисляется разность между суммами. Полученная разность – это значение определенного признака, определенного размера, определенным образом расположенного на изображении.

Каскад Хаара является эффективным способом обнаружения объектов на изображении, он может очень быстро выполнять задачу классификации и иметь устойчивость к различным отклонениям [1].

2. Нейронные сети

Часто в задачах распознавания объектов используют нейронные сети. Нейронная сеть – это математическая модель, построенная по принципу организации и функционирования биологических нейронных сетей – сетей нервных клеток живого организма. Нейронные сети имеют способность анализировать и запоминать различную информацию.

Нейронная сеть имеет вычислительную единицу – нейрон. Сеть состоит из множества нейронов, имеющих свои веса. В самом простом случае нейронная сеть состоит из входного слоя, получающего информацию, несколько скрытых слоев и выходного слоя, выводящего результат. Входная

информация – это сумма всех входных данных, умноженных на соответствующие им веса. Эти данные нормализуются и поступают на следующий слой. Это повторяется до тех пор, пока информация не дойдет до последнего нейрона.

На основе входных данных нейронные сети обучаются и на совершенных ошибках строят свой опыт. В отличие от предыдущего метода, с помощью нейронной сети можно распознавать объекты нескольких классов (более двух) [5].

3. Метод опорных векторов

Это еще один метод классификации, который можно использовать при решении задач детектирования, при использовании определенных признаков. Методом опорных векторов можно определить к какому классу из, как минимум, двух известных относится искомый объект. Суть алгоритма в том, чтобы в процессе тренировки найти наилучшую линию или гиперплоскость, разделяющую данные на два или более класса. К примеру, мы имеем исходные данные в виде точек, относящихся к двум разным классам. Задача алгоритма – найти точки на графике (опорные вектора), расположенные непосредственно к линии разделения ближе всего, затем вычислить расстояние между точками и разделяющей линией и это расстояние максимизировать. Таким образом алгоритм находит наиболее правильную линию или гиперплоскость, разделяющую наши исходные на два класса [6].

В разрабатываемой системе для обнаружения знаков дорожного движения и светофоров будет использоваться каскад Хаара, так как этот метод машинного обучения не требователен к аппаратному обеспечению, является одним из простейших способов распознавания объектов, имея при этом высокую скорость работы и большую точность распознавания.

Однако такой метод подходит для жестких объектов, имеющих ясные отличительные признаки. Поэтому для распознавания дорожной разметки использовался подход, основанный на особенностях цвета и формы объекта.

Для решения поставленной задачи потребуется библиотека OpenCV, поддерживающая работу с компьютерным зрением и языком программирования Python. Данная библиотека является бесплатной и может использоваться как в учебных, так и в коммерческих целях.

3 Разработка структуры системы поиска знаков дорожного движения, светофоров и дорожной разметки

3.1 Общая структура разрабатываемой системы

Общий алгоритм работы разрабатываемой системы представлен на рисунке 3.1. На вход алгоритму подается изображение (или очередной кадр из видеопотока), на котором происходит поиск светофора, дорожной разметки и знаков дорожного движения. Далее алгоритм принимает решение о найденных объектах (какие объекты являются искомыми, а какие нет) и в зависимости от найденного объекта подает управляющие сигналы мобильному роботу.

В данной работе реализована часть алгоритма, выделенная зеленым цветом на рисунке 3.1.

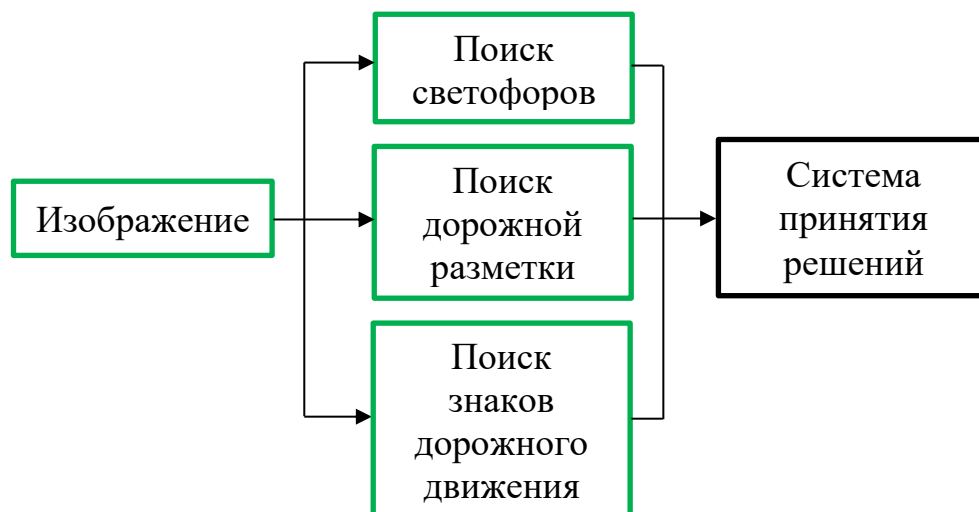


Рисунок 3.1. – Структура разрабатываемой системы

3.1.1 Алгоритм распознавания знаков дорожного движения и светофоров

Программная система распознавания знаков дорожного движения и светофоров включает в себя два этапа:

1. Обучение классификатора с использованием обучающей выборки;
2. Классификация входных данных на основе обученного классификатора.

Перед тем, как обучать классификатор, нужно выполнить следующие пункты:

1. Создать набор данных, необходимых для обучения;
2. Провести предварительную обработку данных.

Задача классификации входных данных состоит в следующем:

1. Получение входных данных из видеозаписи или видеопотока с внешней камеры;
2. Формирование набора признаков для каждого кадра из видеопотока;
3. Принятие решения об идентификации объектов, на основе сформированных признаков.

Алгоритм работы системы по распознаванию знаков дорожного движения и светофоров представлен на рисунке 3.2.



Рисунок 3.2. – Алгоритм распознавания светофоров и знаков

3.1.2 Алгоритм распознавания дорожной разметки

Алгоритм работы системы по распознаванию дорожной разметки представлен на рисунке 3.3. Алгоритм получает на вход изображение или очередной кадр из видеопотока, после чего производит его обработку. Затем алгоритм ищет все прямые линии на изображении, принимает решение о том, какие линии являются частью дорожной разметки, и отображает результат распознавания.

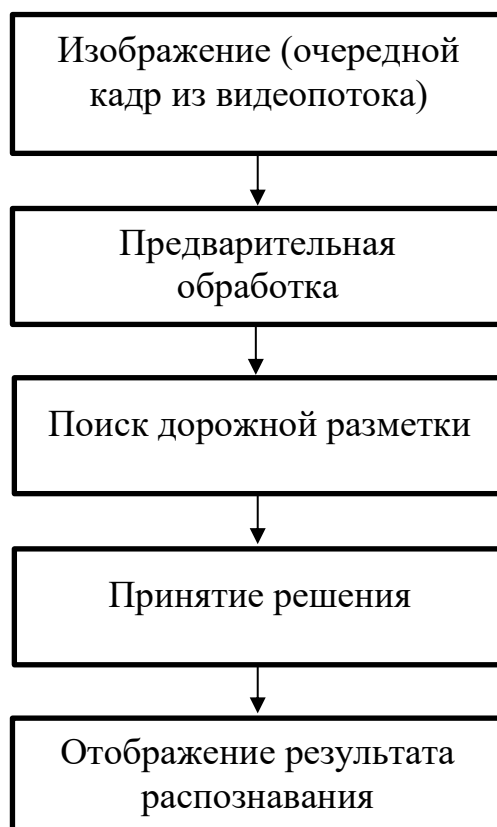


Рисунок 3.3. – Алгоритм распознавания дорожной разметки

3.2 Создание набора данных для обучения каскада Хаара

Данный пункт относится к задаче обнаружения светофоров и знаков дорожного движения. Обучение классификатора будет проводится для светофора с красным и зеленым сигналами и для шести знаков дорожного движения:

1. Въезд запрещен;
2. Движение прямо;
3. Движение направо;
4. Движение налево;
5. Движение прямо или направо;
6. Движение прямо или налево.

Перед обучением классификатора требуется предварительно подготовить набор изображений. Необходимо сделать два набора фотографий. Первый набор должен содержать положительные фотографии, то есть фотографии, на которых присутствует объект, необходимый для обнаружения. Второй набор должен содержать отрицательные фотографии, на которых объект обнаружения отсутствует. Все фотографии должны быть сделаны в той среде, где будет проходить распознавание, так как иначе точность распознавания будет низкой [1].

3.2.1 Положительные примеры

Положительные примеры могут быть получены с помощью программы от OpenCV – `opencv_createsamples`. Данная программа может из одного изображения сгенерировать множество похожих изображений путем случайного изменения ориентации объекта на изображении, изменения интенсивности изображения и размещения объекта на произвольных фонах. Однако данный подход работает плохо, поэтому в данной работе он не будет использован.

Положительные примеры могут быть сделаны самостоятельно. Например, можно сделать большое количество фотографий объекта с разных ракурсов и различного расстояния, либо снять видео объекта и сохранить фотографии из видеопотока. В данной работе использовался второй вариант, так как он занимает гораздо меньше времени. Далее из полученных фотографий нужно вырезать только объект распознавания.

После этого необходимо создать текстовый файл, в котором будут описаны все используемые положительные изображения. Информация о файле должна быть в следующем виде:

Good\0.jpg 1 0 0 200 100

В этой строке «Good\0.jpg» – это путь к изображению, «1» - количество положительных объектов на изображении, «0 0 200 100» - координаты прямоугольника на изображении, в котором находится объект.

После того, как набор положительных фотографий собран и описан в файле, нужно воспользоваться программой `opencv_createsamples`, которая извлекает из файла данных положительные примеры, нормализует их и изменяет до указанного размера. Результат сохраняется в выходном векторном файле (.vec). Имея этот файл, можно многократно повторять процедуру обучения с разными параметрами на одних и тех же данных [1].

Размер изображений нужно установить таким образом, чтобы он приблизительно отражал пропорции искомого объекта. Так, для знаков был установлен размер 20*20, а для светофоров – 15*30.

На рисунке 3.4 показаны некоторые положительные примеры для дорожного знака «Движение налево», на рисунке 3.5 – для светофора с зеленым сигналом.



Рисунок 3.4. – Положительные примеры для знака «Движение налево»



Рисунок 3.5. – Положительные примеры для светофора с зеленым сигналом

3.2.2 Отрицательные примеры

Отрицательные изображения могут быть произвольных размеров, но не меньше размера, установленного для положительных изображений. Отрицательные примеры были получены путем снятия видео комнаты, где будет происходить распознавание, и извлечения снимков из видеопотока [1]. В данной работе все отрицательные фотографии были изменены до размера 100*100.

На рисунке 3.6 показаны некоторые отрицательные фотографии.

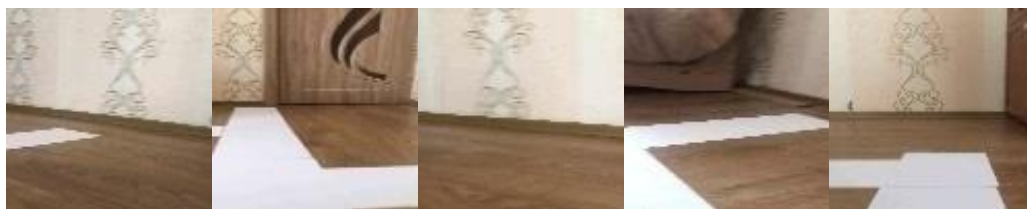


Рисунок 3.6. – Отрицательные примеры

К отрицательным изображениям нужно подмешивать фотографии тех объектов, которые не планируется распознавать в данный момент. Например, если мы собираемся обучить классификатор распознавать красный сигнал светофора, то к отрицательным изображениям нужно добавить фотографии светофора с зеленым сигналом и знаки дорожного движения. Это особенно важно сделать в случае с распознаванием похожих знаков дорожного движения, например, таких как «Движение направо» и «Движение прямо и направо», так как иначе классификатор данные знаки будет путать.

Для отрицательных примеров также необходимо создать текстовый файл с описанием изображений. В данном случае файл должен содержать просто список путей к изображениям.

3.3 Каскад Хаара

Впервые алгоритм обнаружения объектов в реальном времени с помощью признаков Хаара был предложен в статье в 2001 году, авторами которой являются Пола Виола и Майкл Джонс.

Алгоритм Виолы-Джонса имеет следующие ключевые принципы:

1. Признаки Хаара;
2. Интегральное представление изображений;
3. Сканирующее окно;
4. Алгоритм Adaboost [6].

3.3.1 Признаки Хаара

Признаки Хаара имеют прямоугольную (квадратную) форму и состоят из смежных прямоугольных областей. Для вычисления признака Хаара для конкретного изображения, нужно сложить яркости пикселей изображения в первой и второй группах прямоугольных областей по отдельности, а затем вычислить разность между суммами. Разница, которую мы получим будет являться значением конкретного признака Хаара для исследуемого изображения – формула (3.1).

$$\begin{aligned} a_i &= \sum_{j=y_{a_i}}^{y_{a_i}+h_{a_i}-1} \sum_{k=x_{a_i}}^{x_{a_i}+\omega_{a_i}-1} v_{jk} & b_i &= \sum_{j=y_{b_i}}^{y_{b_i}+h_{b_i}-1} \sum_{k=x_{b_i}}^{x_{b_i}+\omega_{b_i}-1} v_{jk} \\ h(u) &= \sum_{i=1}^{N_a} a_i - \sum_{i=1}^{N_b} b_i \end{aligned} \quad (3.1)$$

В формуле (3.1) v_{jk} – яркость пикселя с координатами $[j;k]$, a_i – сумма яркостей пикселей в i -й области первой группы, b_i – сумма яркостей пикселей в i -й области второй группы, h – значение признака Хаара для изображения, $h_{a_i}, \omega_{a_i}, h_{b_i}, \omega_{b_i}$ – высота и ширина i -х областей первой и второй групп соответственно, y_{a_i} и x_{a_i} , y_{b_i} и x_{b_i} – смещения по оси y и x i -х областей первой и второй групп, а N_a и N_b – количество областей в первой и второй группах [1, 5].

На рисунке 3.7 представлены примитивы Хаара, которые используются в стандартном методе Виолы-Джонса.

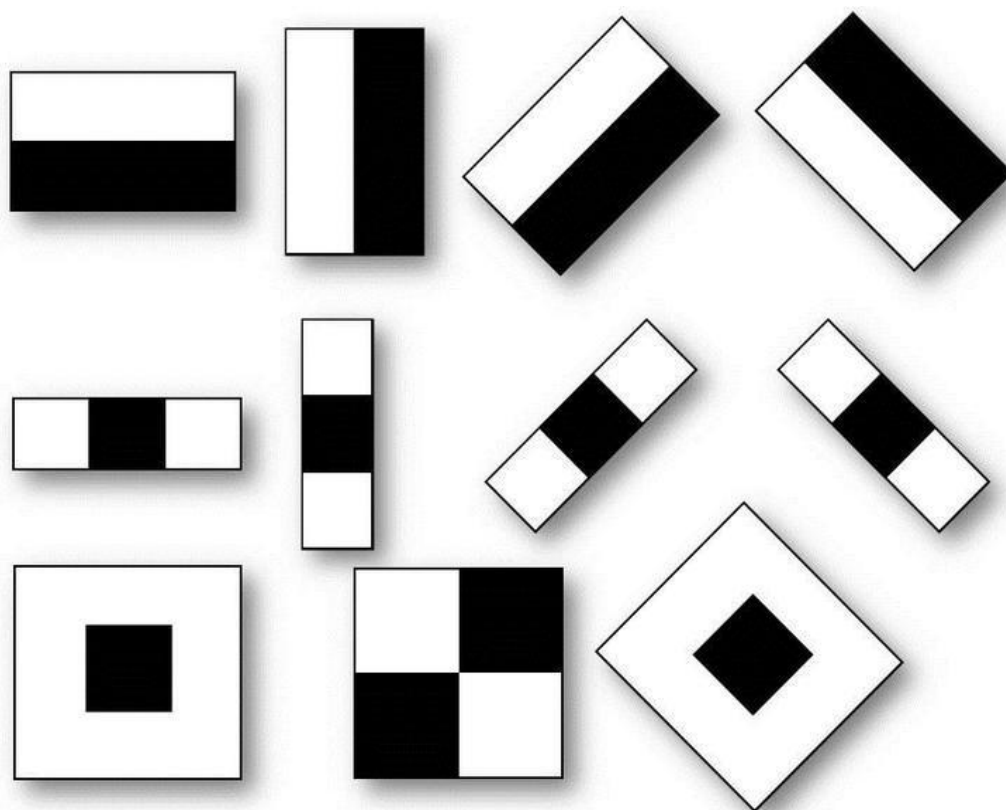


Рисунок 3.7. – Примитивы Хаара, используемые в стандартном методе Виолы-Джонса

В библиотеке OpenCV используется расширенный метод Виолы-Джонса, использующий дополнительные признаки, изображенный на рис. 3.8 [6].

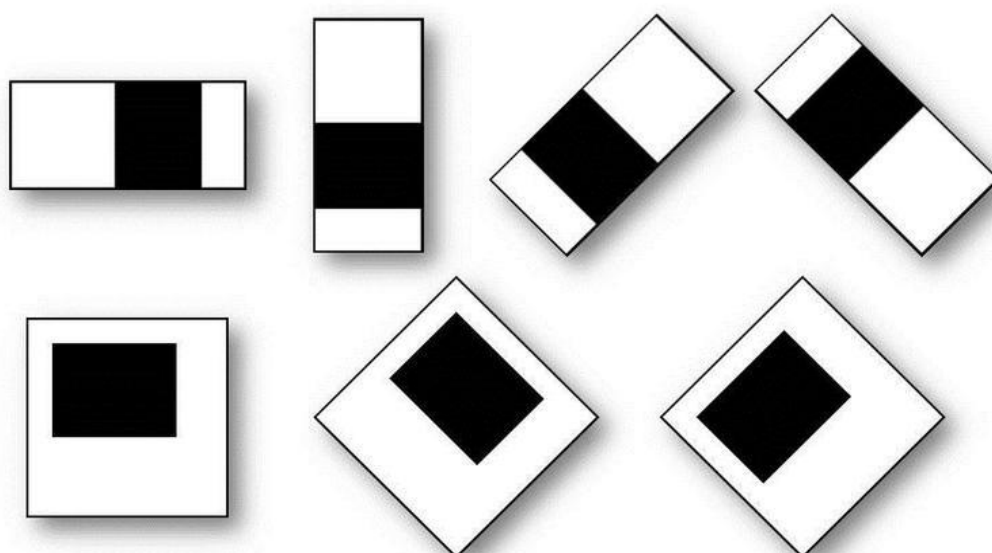


Рисунок 3.8. – Дополнительные примитивы Хаара, используемые в расширенном методе Виолы-Джонса

3.3.2 Интегральное представление изображений

Для вычисления каждого признака Хаара для изображения потребуется большое количество операций сложения, которые займут очень много времени. Поэтому используется интегральная форма представления изображений, так как благодаря этому количество вычислений становится гораздо меньше.

Интегральное представление изображения – это матрица, размер которой больше размера исследуемого изображения на 1 в каждом направлении. Значение каждого элемента в этой матрице – это сумма яркостей данного пикселя и всех пикселей, которые находятся выше и левее него (считаем, что начало координат находится в верхнем левом углу). Таким образом значение пикселя изображения, представленного в интегральной форме, находится по формуле (3.2).

$$S(x, y) = \sum_{i=0}^x \sum_{j=0}^y I(i, j) \quad (3.2)$$

В формуле (3.2) $S(x, y)$ – значение пикселя изображения с координатам $[x; y]$, представленного в интегральной форме, $I(i, j)$ – значение пикселя исходного изображения с координатами $[i; j]$.

Для вычисления интегрального изображения $S(x, y)$ нужно воспользоваться рекуррентной формулой (3.3). Последний член в этой формуле вычитается дважды, так как он дважды учтен: во втором и в третьем членах.

$$S(x, y) = I(x, y) + S(x - 1, y) + S(x, y - 1) - S(x - 1, y - 1) \quad (3.3)$$

Для того, чтобы найти сумму яркостей пикселей в произвольной прямоугольной области, нужно воспользоваться формулой (3.4) [1, 6].

$$\sum_{y_1 \leq y < y_2} \sum_{x_1 \leq x < x_2} I(x, y) = S(x_2, y_2) - S(x_1, y_2) - S(x_2, y_1) + S(x_1, y_1) \quad (3.4)$$

Рассмотрим для примера изображение размером 5*5, представленное на рисунке 3.9 и соответствующее ему интегральное изображение, размером 6*6, представленное на рисунке 3.10.

1	2	5	1	2
2	20	50	20	5
5	50	100	50	2
2	20	50	20	1
1	5	25	1	2

Рисунок 3.9. – Исходное изображение, представленное в числовом виде

0	0	0	0	0	0
0	1	3	8	9	11
0	3	25	80	101	108
0	8	80	235	306	315
0	10	102	307	398	408
0	11	108	338	430	442

Рисунок 3.10. – Изображение в интегральной форме, представленное в числовом виде

Для того чтобы найти сумму яркостей пикселей прямоугольной области исходного изображения со значениями 20 во всех вершинах прямоугольника, нужно вычислить выражение $398-10-9+1 = 380$. Это же число получится при сложении каждого элемента матрицы в рассматриваемом прямоугольнике исходного изображения: $20+50+20+50+100+50+20+50+20 = 380$. Таким образом, используя интегральное представление изображения мы используем всего 4 арифметических действия для вычисления суммы по любому прямоугольнику. Очевидно, что количество операций сложения в этом случае намного сокращается.

3.3.3 Сканирующее окно

В режиме выполнения по изображению передвигается окно сканирования переменного размера. Окно двигается по изображению с шагом в 1 ячейку окна. Сначала сканирование производится с помощью окна минимального размера, установленного вручную, затем размер окна увеличивается (параметр увеличения окна также задается вручную). В каждом окне проверяется около 100000 признаков на всех положительных и отрицательных примерах. Найденные признаки передаются классификатору, который принимает решение [6].

3.3.4 Алгоритм Adaboost

В детекторе Виолы-Джонса используется алгоритм Adaboost, в более широком контексте применяется термин каскад отклонения. Суть алгоритма состоит в том, чтобы из большого количества слабых классификаторов построить новый усиленный классификатор, который бы выполнял задачу распознавания объектов намного эффективнее.

Каскад отклонения состоит из последовательности узлов, представляющих набор слабых классификаторов. Сами узлы являются усиленными классификаторами, а слабые классификаторы – это решающие деревья, которые принимают на вход изображение и сравнивают значение u

некоторого признака Хаара h , вычисленного для данного изображения, с порогом t , возвращая 1 или -1. Ответ «1» означает, что искомый объект присутствует на изображении, «-1» - отсутствует (формула (3.5)).

$$h_m = \begin{cases} +1 & u_m \geq t_m \\ -1 & u_m < t_m \end{cases} \quad (3.5)$$

Далее алгоритм строит узел сильного классификатора в виде взвешенной суммы слабых классификаторов. Функция классификации, используемая в детекторе Виолы-Джонса, представлена в формуле (3.6).

$$H = \text{sign}(\alpha_1 h_1 + \alpha_2 h_2 + \dots + \alpha_{N_m} h_{N_m}) \quad (3.6)$$

Функция sign возвращает -1, если аргумент меньше 0, 0 – если равен нулю, и +1 – если больше нуля. На первом проходе по набору данных для каждого признака h_m обучается порог t_m , который классифицирует входные данные лучше всего. Далее алгоритм усиления использует получившиеся в результате ошибки, чтобы вычислить вес голоса данного классификатора α_m . Вес каждого вектора признаков понижается или повышается в зависимости от того, правильно ли он был классифицирован на данной итерации классификатора. После обучения одного узла данные, уцелевшие после прохождения предыдущих уровней каскада, используются для обучения следующего уровня и т.д [1].

3.4 Описание алгоритма распознавания дорожной разметки

В данной работе был сделан небольшой участок дороги из белых листов бумаги со сплошной и прерывистой черными полосами (рисунок 3.11), которые алгоритм должен уметь распознавать.



Рисунок 3.11. – Исходное изображение

Рассмотрим подробнее каждый этап распознавания дорожной разметки.

3.4.1 Получение и предобработка изображения

Окружающая среда, освещение и плохое качество изображения могут вызвать шум, который может привести к размытым краям линий и плохо повлиять на обнаружение дорожной разметки. Поэтому первым делом необходимо обработать полученное изображение.

Наиболее известное цветовое пространство, в котором хранятся изображения – RGB. В нем каждому каналу присваивается свой цвет – красный (R), зеленый (G) и синий (B). На каждый параметр RGB выделяется 1 байт информации, т. е. каждый пиксель в изображении может принимать значение от 0 до 255. Алгоритмы, которые используются для поиска линий, принимают на вход одноканальное изображение, поэтому цветное изображение следует перевести в оттенки серого. В этом формате изображение будет иметь 1 канал со значениями пикселей, изменяющимися в диапазоне от 0 до 255 [1].

Далее для того чтобы сгладить шум, присутствующий на изображении, к изображению применяется фильтр Гаусса. Фильтр – это алгоритм, который вычисляет новые значения пикселей исходного изображения, учитывая значения окружающих его пикселей. Вычисление новых значений происходит следующим образом: каждый фрагмент изображения поочередно умножается на ядро, затем результат суммируется и это повторяется для всех пикселей исходного изображения. Ядро – матрица определенного размера, заполненная определенным образом.

Для расчета ядра Гаусса используется Гауссова функция (гауссиана) – формула (3.7), где

$$g = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (3.7)$$

σ – стандартное отклонение, влияющее на степень сглаживания (чем выше значение, тем сильнее сглаживание), x и y – координаты пикселя [5].



Рисунок 3.12. – Изображение после предварительной обработки

На рисунке 3.12 представлено изображение после всех обработок. В данных условиях (рис. 3.11) нет сильных помех от окружающей среды и разметка ярко выражена, поэтому такой предварительной обработки будет достаточно.

3.4.2 Поиск дорожной разметки

1. Детектор границ Кэнни

Перед тем как искать линии разметки на изображении, необходимо к изображению применить детектор границ Кэнни, который позволяет найти все границы на изображении и отбросить лишнюю информацию.

Первым этапом работы детектора границ Кэнни является сглаживание изображения для удаления шумов. Здесь применяется фильтр на основе первой производной от гауссианы.

Далее вычисляются градиенты, показывающие направление наискорейшего возрастания некоторой величины (в данном случае функции яркости изображения). Угол направления вектора градиента может принимать следующие значения: 0, 45, 90, 135. Точки, в которых градиенты достигают локального максимума, считаются кандидатами на включение в границы.

Затем нужно определить являются ли найденные точки границей или нет. Для этого к пикселям применяется два порога – верхний и нижний. Если градиент в пикселе больше верхнего порога, то считается, что пиксель принадлежит границе, если меньше нижнего, то пиксель отклоняется. Если градиент находится между порогами, то пиксель не отклоняется только в том случае, если он связан с пикселем, в котором градиент больше верхнего порога.

Градиент может быть вычислен по одной из приведенных ниже формул (3.8), (3.9), где I – это изображение.

$$|grad(x, y)| = \sqrt{\left(\frac{dI}{dx}\right)^2 + \left(\frac{dI}{dy}\right)^2} \quad (3.8)$$

$$|grad(x, y)| = \left|\frac{dI}{dx}\right| + \left|\frac{dI}{dy}\right| \quad (3.9)$$

Вторая формула является более быстрой, но менее точной [1, 2]. На рисунке 3.13 представлено изображение, к которому применен детектор границ Кэнни.

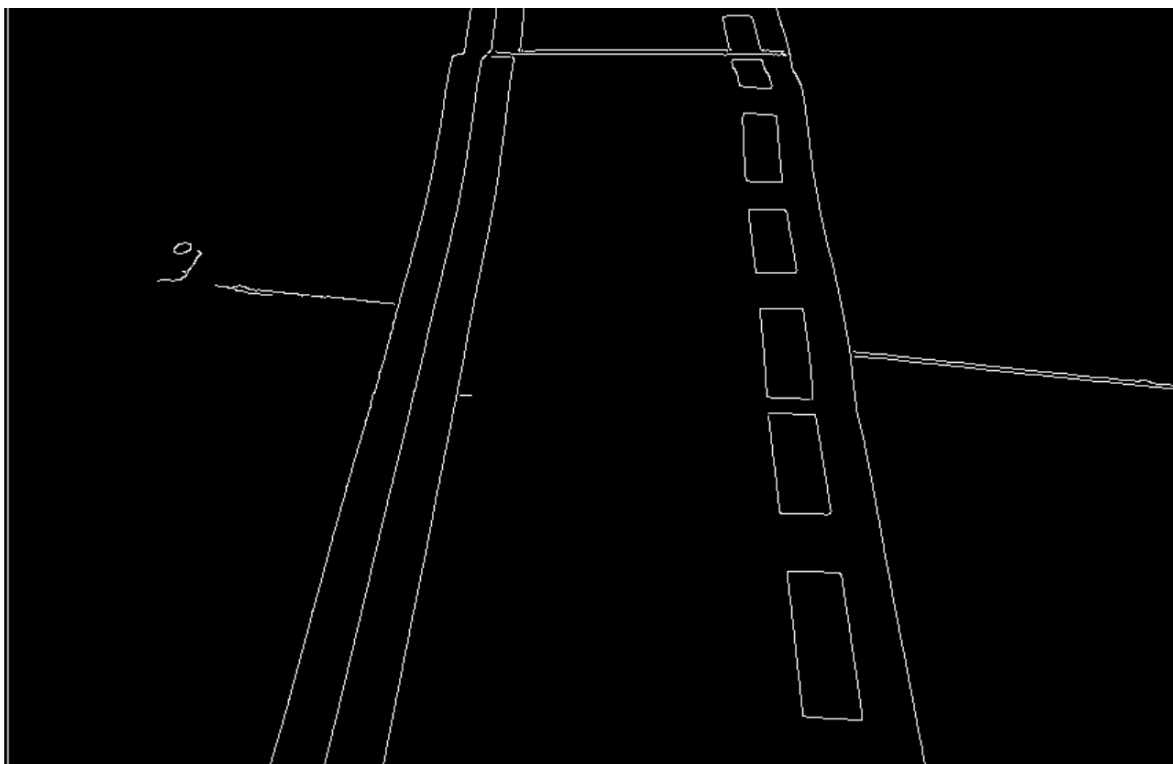


Рисунок 3.13. – Изображение, к которому применен детектор границ Кэнни

2. Выделение области интереса

Данный этап играет важную роль в процессе поиска прямых линий, так как позволяет сразу убрать не интересующие нас области и снизить время процесса поиска полос движения [2].

На рисунке 3.14 область интереса находится внутри черного четырехугольника. Эта область будет сохранена, а все что за ее пределами будет закрашено в черный цвет (рисунок 3.15).

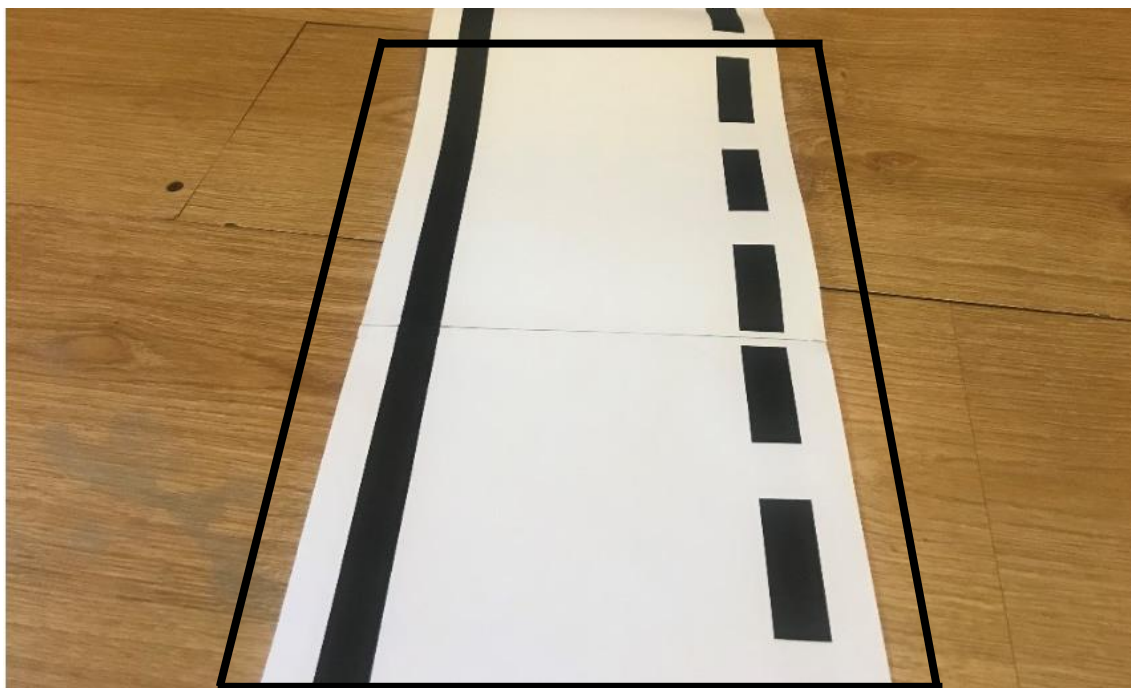


Рисунок 3.14. – Область интереса, находящаяся внутри черного четырехугольника

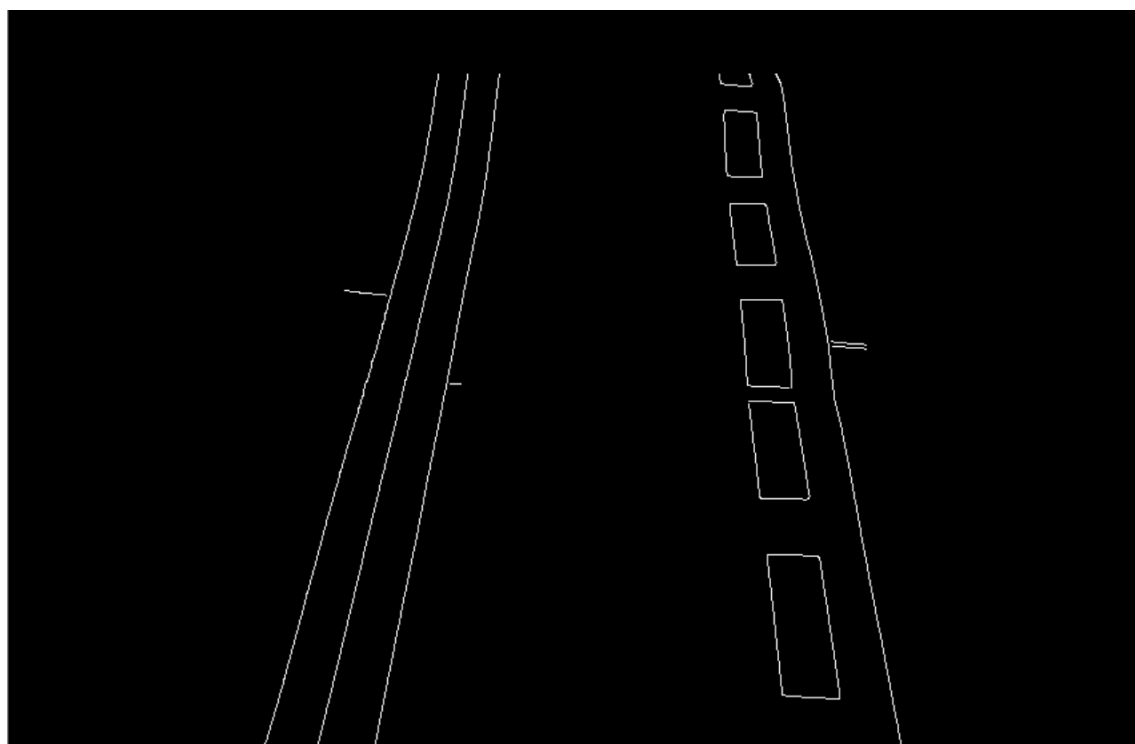


Рисунок 3.15. – Изображение после выделения области интереса

3. Преобразование Хафа

После того, как на изображении найдены все границы и определена область интереса, можно искать на нем прямые линии с помощью преобразования Хафа.

Суть преобразования Хафа в том, что любая точка бинарного изображения может принадлежать некоторому набору возможных прямых линий. Прямая на плоскости описывается уравнением (3.10) и может быть задана парой несовпадающих точек. Однако через одну точку декартовой плоскости можно провести бесконечное множество прямых, поэтому при численных расчетах прямая линия представляется точкой (ρ, θ) в полярной системе координат, так что прямая представлена некоторой точкой проходит через нее перпендикулярно ее радиусу-вектору. Прямая описывается уравнением (3.11).

$$y = kx + b \quad (3.10)$$

$$\rho = x \cos \theta + y \sin \theta \quad (3.11)$$

В формуле (3.11) ρ – длина перпендикуляра, опущенного на прямую из начала координат, θ – угол между перпендикуляром к прямой и осью ОХ (в пределах от 0 до 2π).

Плоскость (ρ, θ) называют аккумуляторной плоскостью или пространством Хафа.

Таким образом с каждой прямой на исходном изображении можно связать точку с координатами (ρ, θ) в пространстве Хафа. Если точка имеет координаты (x_0, y_0) , то прямые, которые проходят через нее, соответствуют уравнению (3.12).

$$\rho = x_0 \cos \theta + y_0 \sin \theta \quad (3.12)$$

Это соответствует синусоидальной линии в аккумуляторной плоскости (ρ, θ) , которая для данной точки уникальна и однозначно ее определяет. Если эти кривые, соответствующие двум точкам, накладываются друг на друга, то точка в пространстве Хафа, где линии пересекаются соответствует прямой в декартовой плоскости, проходящим через обе точки [1, 2, 4].

3.4.3 Принятие решения и отображение результата

После предыдущего шага был сформирован массив, содержащий координаты всех найденных прямых линий.

В первую очередь, из массива линий нужно сформировать два новых массива, один из которых будет содержать координаты левой полосы движения, а другой – правой. Если отдельная точка из массива линий по оси ОХ находится левее середины изображения, то считается, что она принадлежит к левой полосе движения, иначе - к правой [2].

Далее нужно провести аппроксимацию найденных линий.

					ВКР-НГТУ-09.03.01-(16-В-1)-017-2020 (ПЗ)	Лист
						31
Изм.	Лист	№ докум.	Подп.	Дата		

Аппроксимация – это научный метод, основанный на замене одних математических объектов другими, более простыми, но в каком-то смысле близкими к исходным. Пусть имеется некоторый набор данных, в нашем случае это точки, описывающие прямые, с координатами (x_i, y_i) . Аппроксимация – это метод приближения, при котором происходит построение математической функции или кривой, которая не обязательно проходит через исходные точки, а наилучшим образом приближается к ним.

Чтобы построить такую функцию можно воспользоваться двумя способами:

1. Построение интерполяционного многочлена n -й степени, проходящего непосредственно через все исходные точки;
2. Построение аппроксимирующего многочлена n -й степени, проходящего вблизи от исходных точек. Аппроксимирующая функция таким образом сглаживает все случайные помехи, которые могли возникнуть в ходе получения исходных данных [5, 6].

На рисунке 3.16 синяя линия является результатом аппроксимации, а зеленая – интерполяции. Красные маркеры – это исходный набор данных.

В данной работе используется второй вариант построения функции, которая определяется по методу наименьших квадратов.

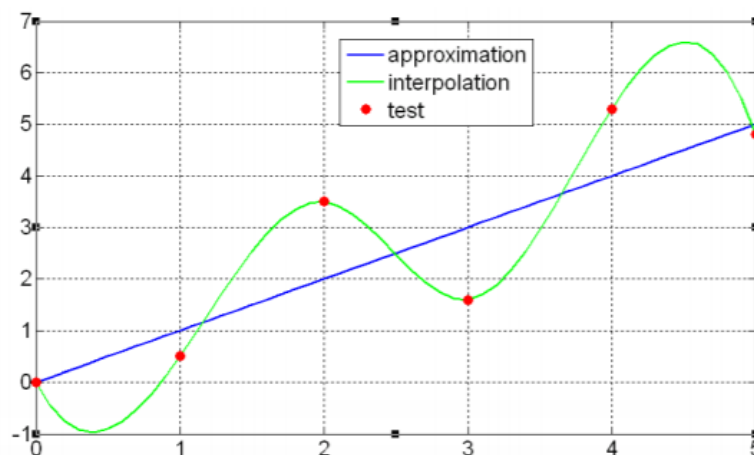


Рисунок 3.16. – Исходный набор данных и построенные функции аппроксимации и интерполяции

Метод наименьших квадратов – это математический метод, суть которого – найти аппроксимирующую функцию, которая строится вблизи от исходных точек. Близость исходной и аппроксимирующей функции определяется числовой мерой: сумма квадратов отклонений исходных данных от аппроксимирующей кривой должна быть наименьшей – формула (3.13), где

$$\sum_{i=1}^N \sigma_i^2 = \sum_{i=1}^N (F(x_i) - y_i)^2 \rightarrow \min \quad (3.13)$$

$F(x_i)$ – значения расчетной аппроксимирующей функции в точках x_i ,
 y_i – заданный массив данных в точках x_i .

Аппроксимирующая функция представляет собой многочлен степени m – формула 3.14, где

$$F_m(x) = a_0 + a_1x + \dots + a_{m-1}x^{m-1} + a_mx^m \quad (3.14)$$

a_m – неизвестные коэффициенты аппроксимирующего многочлена степени m , N – количество заданных значений.

Если степень многочлена $m = 1$, то исходная функция аппроксимируется линией (линейная регрессия), если $m = 2$, то квадратичной параболой, если $m = 3$, то кубической параболой. В данной работе производилась линейная аппроксимация [5].

Аппроксимация в python осуществляется с помощью функции *polifit()*. Функция $p = \text{polifit}(x, y, m)$ находит коэффициенты полинома степени m , который аппроксимирует функцию $y(x)$ методом наименьших квадратов. А для того, чтобы построить полином в python используется функция *polyld()*, которой в качестве аргументов передаются коэффициенты полинома.

Прямая линия строится с помощью функции *cv2.line()* по двум точкам: первая точка определяет начало линии, а вторая – ее конец [3].

Для наилучшего отображения результата линии нужно продлить до конца изображения, то есть заменить значение точек, которые являются концами линий, по оси ОУ на значение равное высоте изображения. Также нужно сделать так, чтобы линии начинались на одном уровне. Для этого сравниваются две точки по оси ОУ, которые являются началом линий. Из этих значений выбирается минимальное и к нему приравниваются обе точки.

4 Разработка программных средств

4.1 Программная реализация алгоритма распознавания светофоров и знаков дорожного движения

Первым делом нужно создать набор данных для обучения. Как было сказано в пунктах 3.2.1 и 3.2.2, все фотографии должны быть получены из видеопотока. Для этого была создана функция `get_img_from_video_file()` – рисунок 4.1.

```
# Получение изображений из заранее снятого видео
def get_img_from_video_file():
    # Создание объекта VideoCapture и чтение из входного файла
    cap = cv2.VideoCapture(video_file)
    # Проверка на успешное открытие файла
    if (cap.isOpened() == False):
        print("Error opening video file")
    # Чтение каждого кадра, пока видео не закончено
    i = 0
    while (cap.isOpened()):
        # Захват по кадрам
        ret, frame = cap.read()
        i += 1
        if ret == True:
            cv2.imwrite(os.path.join(GOOD_FOLDER, str(i) + '.jpg'), frame)
            # Показать полученный кадр
            cv2.imshow('Frame', frame)
            # Пока не нажата клавиша Q на клавиатуре, видео будет продолжать воспроизводиться
            if cv2.waitKey(25) & 0xFF == ord('q'):
                break
        else:
            break
    # Очистить все окна и освободить память
    cap.release()
    cv2.destroyAllWindows()
```

Рисунок 4.1. – Получение набора фотографий из видеопотока

Далее из полученных положительных фотографий нужно вырезать только объект распознавания, то есть светофоры и знаки дорожного движения. Для этого был создан модуль (рисунок 4.2), который позволяет с помощью мышки выделять область на изображении, которую необходимо вырезать. Вырезанная область сохраняется вместо исходного изображения.

					ВКР-НГТУ-09.03.01-(16-В-1)-017-2020 (ПЗ)	Лист
						34
Изм.	Лист	№ докум.	Подп.	Дата		

```

def click_and_crop(event, x, y):
    # глобальные переменные
    global refPt, cropping
    # если была нажата левая кнопка мыши, то начальные
    # координаты (x, y) записываются и указывается, что выполняется обрезка
    if event == cv2.EVENT_LBUTTONDOWN:
        refPt = [(x, y)]
        cropping = True
    # проверка на то, что левая кнопка мыши была отпущена
    elif event == cv2.EVENT_LBUTTONUP:
        # запись конечных координат (x, y) и указание того, что обрезка завершена
        refPt.append((x, y))
        cropping = False
        # рисование прямоугольника вокруг выделенной области
        cv2.rectangle(image, refPt[0], refPt[1], (0, 255, 0), 2)
        cv2.imshow("image", image)

def crop(image, file):
    # цикл продолжается пока не нажата клавиша
    while True:
        # вывод изображения на экран и ожидание нажатия клавиши
        cv2.imshow("image", image)
        key = cv2.waitKey(1) & 0xFF
        # если нажата клавиша "r", то выделенная область обрезки сбрасывается
        if key == ord("r"):
            image = clone.copy()
        # при нажатии клавиши "c" происходит выход из цикла
        elif key == ord("c"):
            break
    # если есть 2 опорные точки, то выделенная область обрезается
    # и выводится на экран
    if len(refPt) == 2:
        roi = clone[refPt[0][1]:refPt[1][1], refPt[0][0]:refPt[1][0]]
        cv2.imwrite(os.path.join(GOOD_FOLDER, file), roi)
        cv2.waitKey(0)
    cv2.destroyAllWindows()

```

Рисунок 4.2. – Функции для обрезки изображений

С помощью функций `create_bad_txt()` и `create_good_txt()` создавались текстовые файлы для отрицательных и положительных изображений соответственно (рисунок 4.3).

```

# создание текстового файла с описанием отрицательных изображений
def create_bad_txt():
    for root, dirs, files in os.walk(BAD_FOLDER):
        for file in files:
            file_path = os.path.join(BAD_FOLDER, file)
            # создать файл с описанием отрицательных изображений
            my_file = open(BAD_TXT_FILE, "a")
            my_file.write(file_path + '\n')
            my_file.close()

# создание текстового файла с описанием положительных изображений
def create_good_txt():
    for root, dirs, files in os.walk(GOOD_FOLDER):
        for file in files:
            file_path = os.path.join(GOOD_FOLDER, file)
            img = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
            image_shape = img.shape
            # создать файл с описанием отрицательных изображений
            my_file = open(GOOD_TXT_FILE, "a")
            my_file.write(file_path + ' 1 0 0 ' + str(image_shape[1]) + ' ' + str(image_shape[0]) + '\n')
            my_file.close()

```

Рис. 4.3. – Создание текстовых файлов описаний положительных и отрицательных изображений

Перед тем как обучать классификатор, необходимо правильно подобрать параметры обучения. На рисунке 4.4 показаны параметры для обучения классификатора для одного из знаков дорожного движения.

```

GOOD_SIZE = (20, 20) # 25,45 # Размер шаблона

NUM_STAGES = 14 # количество уровней каскада
MIN_HIT_RATE = 0.999 # коэффициент, определяющий качество обучения
MAX_FALSE_ALARM_RATE = 0.4 # уровень ложной тревоги
NUM_OF_GOOD = 1000 # общее количество имеющихся положительных примеров
NUMGOOD = int(NUM_OF_GOOD * 0.80) # 80% от имеющихся положительных примеров
NUMBAD = 3200 # общее количество имеющихся отрицательных примеров
MODE = 'ALL' # использовать полный комплект Хаар-признаков
MEMORY_SIZE = 1024 # выделяемая под процесс память

```

Рисунок 4.4. – Параметры для обучения классификатора для знака «Движение налево»

Как было сказано в пункте 3.2.1 перед обучением нужно создать векторный файл (.vec) с помощью программы `opencv_createsamples`. Для этого используется функция `get_vector_file()` – рисунок 4.5.

```

# создание пачки приведённых положительных изображений
def get_vector_file():
    subprocess.check_call(["D:\\opencv\\build\\x64\\vc12\\bin\\opencv_createsamples",
                           "-info", GOOD_TXT_FILE,
                           "-vec", VEC_FILE,
                           "-w", str(GOOD_SIZE[0]),
                           "-h", str(GOOD_SIZE[1])])

```

Рисунок 4.5. – Создание векторного файла с помощью программы opencv_createsamples

Описание параметров, которые указываются в программе opencv_createsamples:

-info – имя текстового файла, содержащего описание положительных изображений;

-vec – имя векторного файла, создаваемого программой opencv_createsamples;

-w, -h – ширина и высота генерируемых примеров в пикселях [3].

Далее можно приступать к обучению классификатора. Обучение производится с помощью программы opencv_traincascade (рисунок 4.6).

```

# создаем итоговый каскад
def train_haar_cascade():
    if not os.path.exists(DATA_FOLDER):
        os.makedirs(DATA_FOLDER)

    subprocess.check_call(["D:\\opencv\\build\\x64\\vc12\\bin\\opencv_traincascade",
                           "-data", DATA_FOLDER,
                           "-vec", VEC_FILE,
                           "-bg", BAD_TXT_FILE,
                           "-numStages", str(NUM_STAGES),
                           "-minHitRate", str(MIN_HIT_RATE),
                           "-maxFalseAlarmRate", str(MAX_FALSE_ALARM_RATE),
                           "-numPos", str(NUMGOOD),
                           "-numNeg", str(NUMBAD),
                           "-w", str(GOOD_SIZE[0]),
                           "-h", str(GOOD_SIZE[1]),
                           "-mode", MODE,
                           "-precalcValBufSize", str(MEMORY_SIZE),
                           "-precalcIdxBufSize", str(MEMORY_SIZE)])

```

Рисунок 4.6. – Обучение классификатора с помощью программы opencv_traincascade

Описание параметров, которые указываются в программе `opencv_traincascade`:

-data – параметр задает имя файла результирующего обученного классификатора (создается файл формата .xml);

-vec – имя векторного файла с положительными примерами;

-bg – текстовый файл с описание отрицательных примеров;

-numStages – сколько уровней будет содержать каскадный классификатор;

-minHitRate – задает минимальную долю правильно распознанных истинных объектов. Значение принадлежит диапазону от 0 до 1, то есть доле 99.8% соответствует значение 0.998. Это целевой показатель для одного уровня каскада, поэтому окончательный коэффициент обнаружения приблизительно равен этому числу, возведенному в степень, равную числу уровней;

-maxFalseAlarmRate – задает максимальную долю ложноположительных результатов, то есть участков изображения, неправильно распознанных как интересующие нас объекты. Конечно в идеале должно быть 0%, но на самом деле, здесь задается довольно большое значение, так как мы рассчитываем, что каскад отклонит «ложные тревоги»;

-numPos – число, которое указывает сколько положительных примеров использовать для обучения каждого уровня каскада. Обычно указывается число, равное 80% от имеющихся положительных примеров;

-numNeg – число, которое указывает сколько отрицательных примеров использовать для обучения каждого уровня каскада;

-w – ширина примеров, сгенерированных программой `opencv_createsamples`;

-h – высота примеров, сгенерированных программой `opencv_createsamples`;

-mode – определяет какие признаки Хаара использовать: только оригинальные или расширенные;

-precalcValBufSize – размер буфера в МБ, выделенного для хранения ранее вычисленных значений признаков. Данный буфер используется в реализации алгоритма усиления для хранения вычисленных признаков, чтобы не пересчитывать их каждый раз;

-precalcIdxBufSize – размер кэша для «значений буферных индексов». Также используется в реализации алгоритма усиления. Рекомендуется, чтобы это значение было равно `precalcValBufSize` [3].

После обучения каскада необходимо его протестировать. Сначала обученный классификатор нужно загрузить в программу и прочитать видеозапись (рисунок 4.7).

```

if __name__ == '__main__':
    # путь к видеозаписи
    filepath = 'C:\\Users\\Natalia\\Desktop\\video_signs\\IMG_3050.mp4'
    # загрузка обученных классификаторов
    sign1_cascade = cv2.CascadeClassifier(
        'D:\\PythonProjects\\DETECT_TRAFFIC_SIGN\\Haar_cascad\\signs\\kirpich\\1\\data\\kirpich.xml')
    # чтение видеозаписи
    cap = cv2.VideoCapture(filepath)

```

Рисунок 4.7. – Загрузка обученного каскада и чтение видеозаписи

Далее каждый кадр необходимо обработать (обрезать и перевести в черно-белое пространство) и с помощью функции обнаружения detectMultiScale() найти объекты на каждом кадре – рисунок 4.8.

```

while (cap.isOpened()):
    ret, frame = cap.read()
    if (ret):
        # обрезаем каждый кадр
        cropped_frame = crop_frame(frame)
        # переводим кадр в ч/б
        gray_filered = cv2.cvtColor(cropped_frame, cv2.COLOR_BGR2GRAY)
        # функция поиска знаков
        signs_kirpich = sign1_cascade.detectMultiScale(gray_filered, scaleFactor=1.15, minNeighbors=12,
                                                         minSize=(15, 15))

```

Рисунок 4.8. – Обработка каждого кадра и поиск объекта

Функции detectMultiScale() передаются следующие аргументы:

scaleFactor – задает коэффициент перехода от одного масштаба сканирующего окна к другому. Чем выше данное значение, тем быстрее вычисление, но из-за этого могут быть пропуски искомого объекта некоторых размеров;

minNeighbors – предотвращает ложное распознавание. Один и тот же объект обычно обнаруживается несколько раз, потому что соседние пиксели и масштабы свидетельствуют о наличии объекта. Если данное значение равно трем, то существование объекта в некоторой области признается, только если оно обнаружено по меньшей мере три раза при перекрывающихся положениях окна;

minSize, maxSize – данные аргументы задают минимальный и максимальный размеры области, в которой будет производиться поиск объекта [3].

Далее происходит рисование прямоугольника некоторого цвета вокруг найденного объекта, а также подпись сверху о том, что это за объект (рисунок 4.9).

```

    # рисование прямоугольника вокруг найденного объекта
    for (x, y, w, h) in signs_kirpich:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 255, 0), 3)
        cv2.putText(frame, "NO ENTRY", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 0), 2)
    out.write(frame)
    cv2.imshow('frame', frame)
    cv2.waitKey(1)
else:
    break
k = cv2.waitKey(25) & 0xFF
if k == 27:
    break

cap.release()
cv2.destroyAllWindows()

```

Рисунок 4.9. – Рисование рамки вокруг найденного объекта и подпись

4.2 Программная реализация алгоритма распознавания дорожной разметки

Во-первых, необходимо получить изображение (или очередной кадр из видеопотока) и затем его обработать. Для этого использовалась функция `prepare_img()`, которая в том числе применяет к изображению детектор границ Кэнни – рисунок 4.10.

```

def prepare_img(img):
    # перевод изображения в черно-белое пространство
    gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    # применение к изображению фильтра Гаусса
    blur_img = cv2.GaussianBlur(gray_img, (kernel_size, kernel_size), 0)
    # применение к изображению детектора границ Кэнни
    Canny_img = cv2.Canny(np.uint8(blur_img), low_threshold, high_threshold)
    return Canny_img

```

Рисунок 4.10. – Обработка изображения и применение к нему детектора границ Кэнни

Далее нужно выделить область интереса на изображении с помощью функции `region_of_interest()` – рисунок 4.11. В данной функции важно правильно задать все вершины четырехугольника, ограничивающего область интереса (рисунок 3.14).

```

def region_of_interest(img):
    # размер изображения, h-высота, w-ширина
    h = img.shape[0]
    w = img.shape[1]
    # определение вершин для области интереса
    # (левая верхняя точка исходного изображения имеет координаты (0;0),
    # правая нижняя точка исходного изображения имеет координаты (x;y)).
    print(w,h)
    # левая нижняя точка области интереса
    bottom_left = (200, h)
    # левая верхняя точка области интереса
    top_left = (300, 50)
    # правая верхняя точка области интереса
    top_right = (700, 50)
    # правая нижняя точка области интереса
    bottom_right = (750, h)

    # создание массива, определяющего область интереса на изображении
    vertices = np.array([[bottom_left,
                           top_left,
                           top_right,
                           bottom_right]],
                        dtype=np.int32)
    # изображение для рисования (маска)
    image_to_draw = np.zeros_like(img)
    # определения цвета для закрашивания области интереса
    # в зависимости от входного изображения (одноканальное/многоканальное)
    if len(img.shape) > 2:
        channel_count = img.shape[2] # количество каналов у изображения (3 или 4)
        color = (255,) * channel_count
    else:
        color = 255
    # закрашивание полигона, определенного вершинами цветом заливки
    cv2.fillPoly(image_to_draw, vertices, color)
    # наложение маски поверх оригинального изображения, сохраняя каждый пиксель
    # изображения, если соответствующее значение маски равно 1
    image_mask = cv2.bitwise_and(img, image_to_draw)
    return image_mask

```

Рисунок 4.11. – Выделение области интереса на изображении

Затем можно приступить к поиску линий на изображении с помощью функции `search_lines()` – рисунок 4.12.

```

def search_lines(img):
    # преобразование Хафа для поиска прямых линий (возвращает 2 конечные точки линий)
    lines = cv2.HoughLinesP(img, rho, theta, threshold, np.array([]), minLineLength=min_line_length,
                             maxLineGap=max_line_gap)
    # создаем массив такого же размера как исходное изображение, заполненный нулями
    line_img = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)
    return line_img, lines

```

Рисунок 4.12. – Поиск прямых линий на изображении

И наконец, с помощью функции `draw_lines()` результат отображается на экране – найденные линии рисуются поверх исходного изображения (рисунок 4.13).

```
def draw_lines(img, lines, color=(255, 0, 0), thickness=10):
    # изменяем размер массива линий, делая его двумерным
    lines = lines.reshape(lines.shape[0], lines.shape[2])
    # lines - массив линий, линия состоит из двух точек (x1,y1) и (x2,y2)
    # меняем массив линий на массив отдельных точек
    lines = np.reshape(lines, (lines.shape[0] * 2, 2))
    # считаем, что если точка находится справа от середины изображения,
    # то она принадлежит к правой полосе
    right_lines = np.array(list(filter(lambda x: x[0] > (img.shape[1] / 2), lines)))
    # считаем, что если точка находится слева от середины изображения,
    # то она принадлежит к левой полосе
    left_lines = np.array(list(filter(lambda x: x[0] < (img.shape[1] / 2), lines)))
    if len(right_lines) != 0 and len(left_lines) != 0 and len(lines) != 0:
        # находим нижнюю точку правой полосы
        min_right_x, min_right_y = np.amin(right_lines, axis=0)
        # находим нижнюю точку левой полосы
        min_left_x, min_left_y = np.amin(left_lines, axis=0)

        # находим аппроксимирующую функцию для правой и левой полосы по методу наименьших квадратов
        right_curve = np.poly1d(np.polyfit(right_lines[:, 1], right_lines[:, 0], 2))
        left_curve = np.poly1d(np.polyfit(left_lines[:, 1], left_lines[:, 0], 2))

        # Решаем полином, вычисляем максимальное и минимальное значения по x
        max_right_x = int(right_curve(img.shape[0]))
        min_right_x = int(right_curve(min_right_y))

        # Решаем полином, вычисляем значения x
        min_left_x = int(left_curve(img.shape[0]))
        max_left_x = int(left_curve(min_left_y))

        # находим самую низкую точку, для того чтобы полосы
        # начинались на одном уровне. А значение самой высокой точки
        # будет равно значению высоты изображения, для того чтобы линии
        # были отображены до конца изображения
        min_y = min(min_right_y, min_left_y)

        r1 = (min_right_x, min_y)
        r2 = (max_right_x, img.shape[0])
        cv2.line(img, r1, r2, color, thickness)

        l1 = (max_left_x, min_y)
        l2 = (min_left_x, img.shape[0])
        cv2.line(img, l1, l2, color, thickness)
```

Рисунок 4.13. – Рисование найденных линий

5 Тестирование системы

5.1 Тестирование системы распознавания светофоров и знаков дорожного движения

Для обучения каскада Хаара для каждого объекта в среднем было сделано 1000 положительных и 3000 отрицательных фотографий. Изначально было задано 14 уровней каскада, но в процессе обучения для всех объектов на 8 (9) уровне была достигнута заданная скорость ложной тревоги (0.4) и обучение прекратилось. Время обучения каждого каскада составило около четырех часов.

Для эксперимента фотографии знаков и светофора были сделаны с разных ракурсов.

На рисунке 5.1 представлены результаты распознавания дорожного знака «Движение прямо и направо», на рисунке 5.2 – «Движение налево», на рисунке 5.3 – «Движение прямо и налево», на рисунке 5.4 – «Движение запрещено», на рисунке 5.5 – «Движение прямо», на рисунке 5.6 – «Движение направо».



Рисунок 5.1. – Результаты распознавания дорожного знака «Движение прямо и направо»

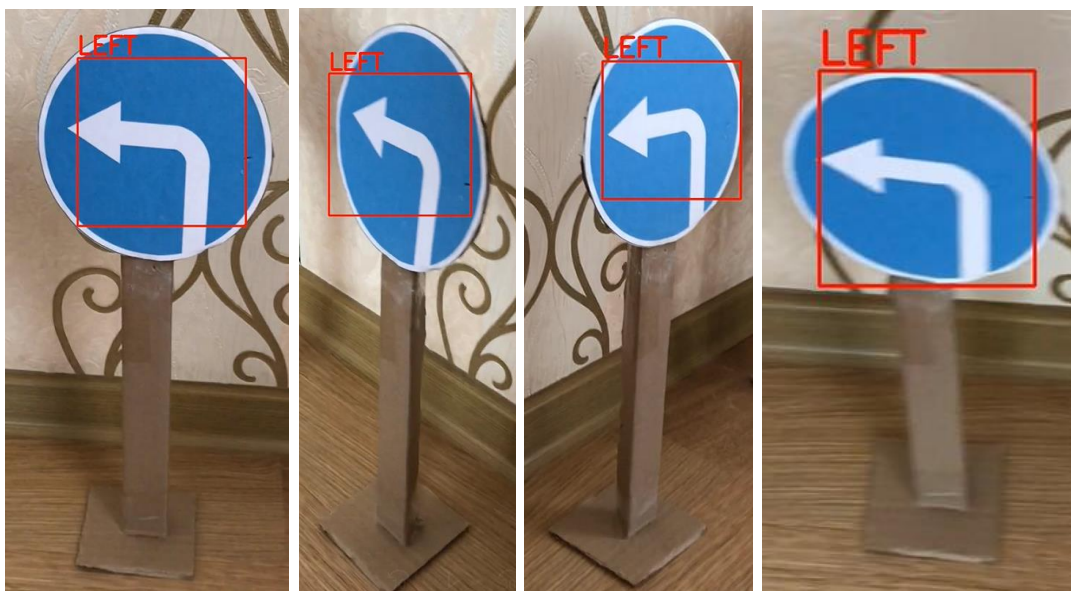


Рисунок 5.2. – Результаты распознавания дорожного знака «Движение налево»

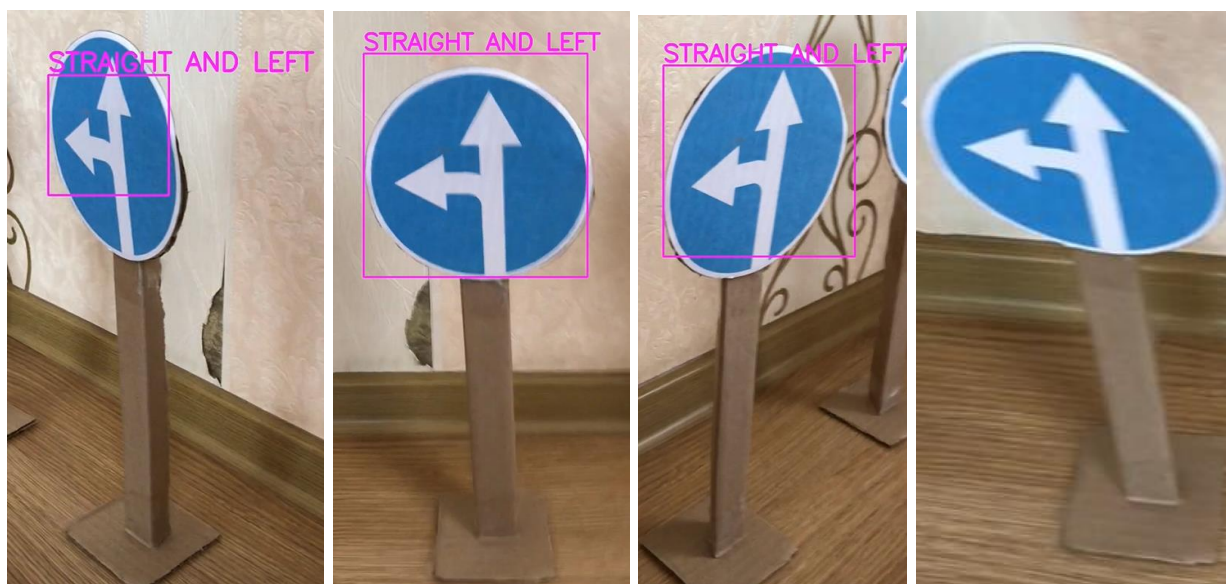


Рисунок 5.3. – Результаты распознавания дорожного знака «Движение прямо и налево»

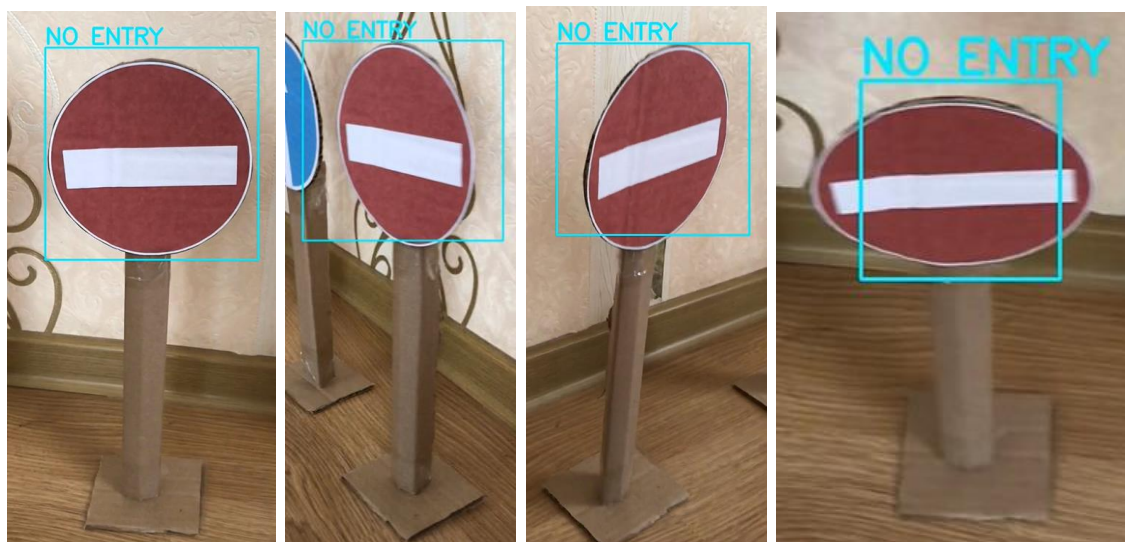


Рисунок 5.4. – Результаты распознавания дорожного знака «Движение запрещено»

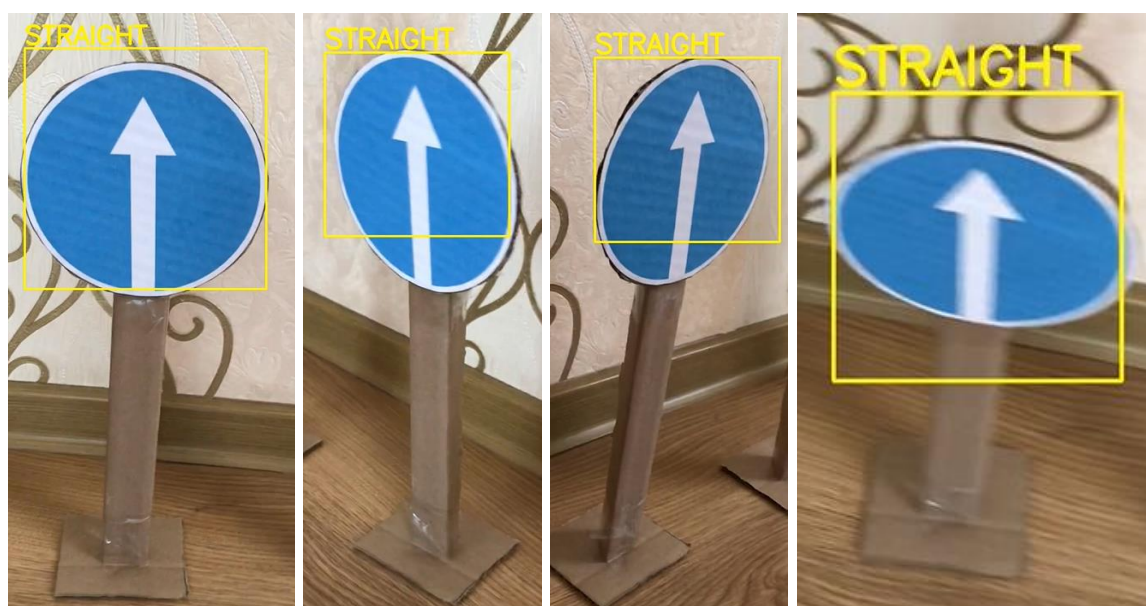


Рисунок 5.5. – Результаты распознавания дорожного знака «Движение прямо»



Рисунок 5.6. – Результаты распознавания дорожного знака «Движение направо»

Результаты тестирования показывают, что знаки распознаются классификатором очень хорошо. Однако с некоторых ракурсов различные знаки не могут быть обнаружены. Например, на рисунке 5.3 видно, что знак «Движение прямо и налево» не распознан с верхнего ракурса, также, как и знак «Движение прямо и направо» - рисунок 5.1. Я думаю, это связано с тем, что для данных знаков недостаточно положительных примеров с такого ракурса.

На рисунке 5.7. показаны результаты распознавания для светофора с зеленым сигналом, а на рисунке 5.8 – для светофора с красным сигналом.

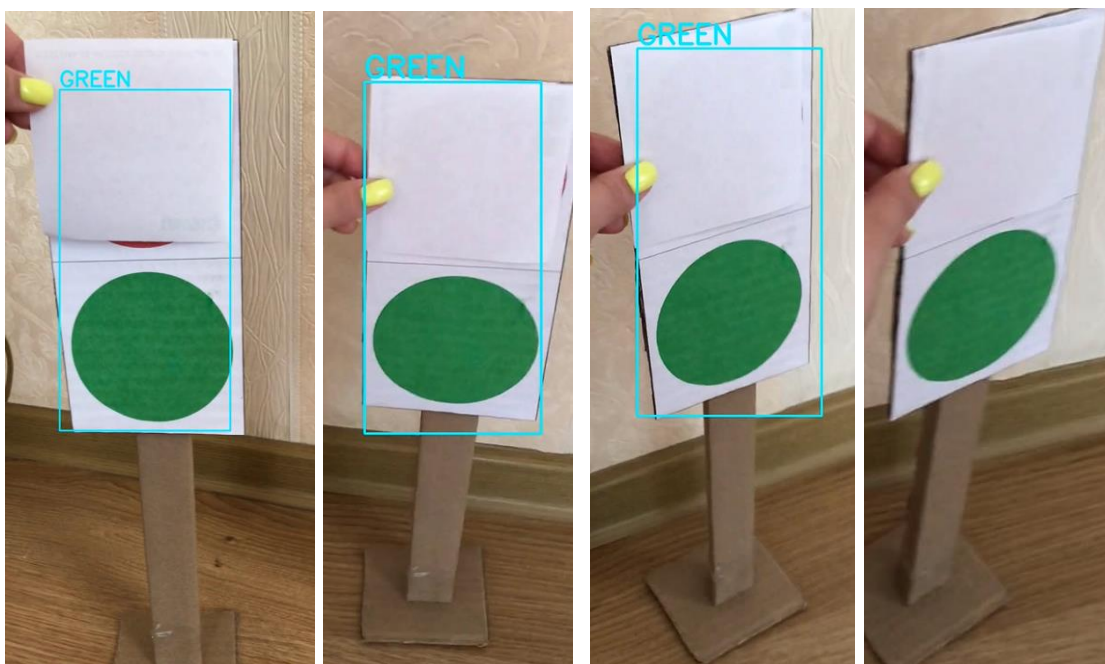


Рисунок 5.7. – Результаты распознавания светофора с зеленым сигналом

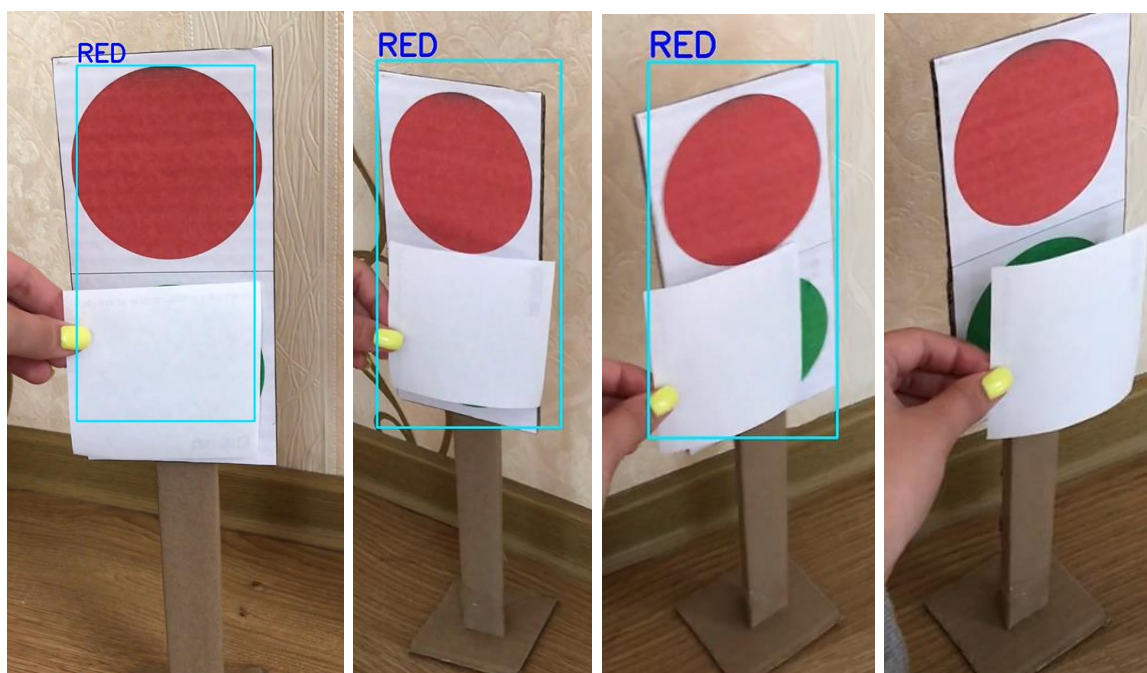


Рисунок 5.8. – Результаты распознавания светофора с красным сигналом

На рисунках 5.7 и 5.8 также видно, что светофор распознается не со всех ракурсов. Если светофор расположен прямо перед камерой или с небольшим смещением точки съемки относительно него, то он будет хорошо распознан классификатором.

Также могут возникнуть ложные срабатывания каскада Хаара (рисунок 5.9). Ложных срабатываний можно избежать или уменьшить их количество, если увеличить значение minNeighbor и подобрать нужные значения minSize и maxSize в функции поиска объектов detectMultiScale().

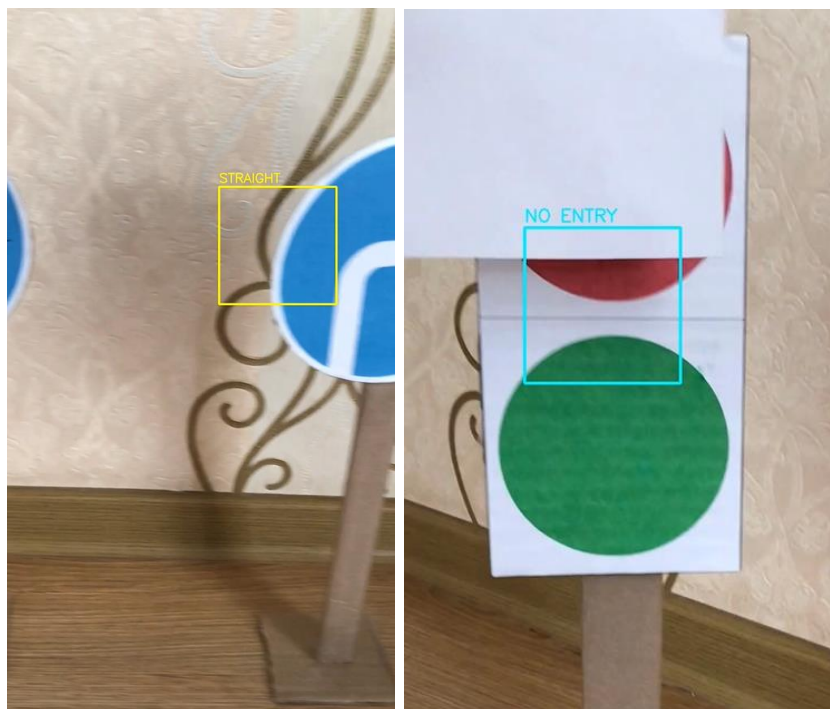


Рисунок 5.9. – Ложные срабатывание классификатора

5.2 Тестирование системы распознавания дорожной разметки

Для тестирования системы было взято 2 изображения дороги, снятых при разном освещении. Также для эксперимента линии на изображении были нарисованы сразу после преобразования Хафа и после их аппроксимации.

На рисунке 5.10 представлено изображение, снятое при обычном комнатном свете и линии на нем нарисованы сразу после преобразования Хафа. На рисунке 5.11 представлено то же самое изображение, но линии на нем нарисованы после их аппроксимации. Из этих рисунков видно, что аппроксимация улучшает результат распознавания.



Рисунок 5.10. – Отображение результата до аппроксимации линий



Рисунок 5.11. – Отображение результата после аппроксимации линий

На рисунке 5.12 показан результат распознавания дорожной разметки при ярком солнечном свете.

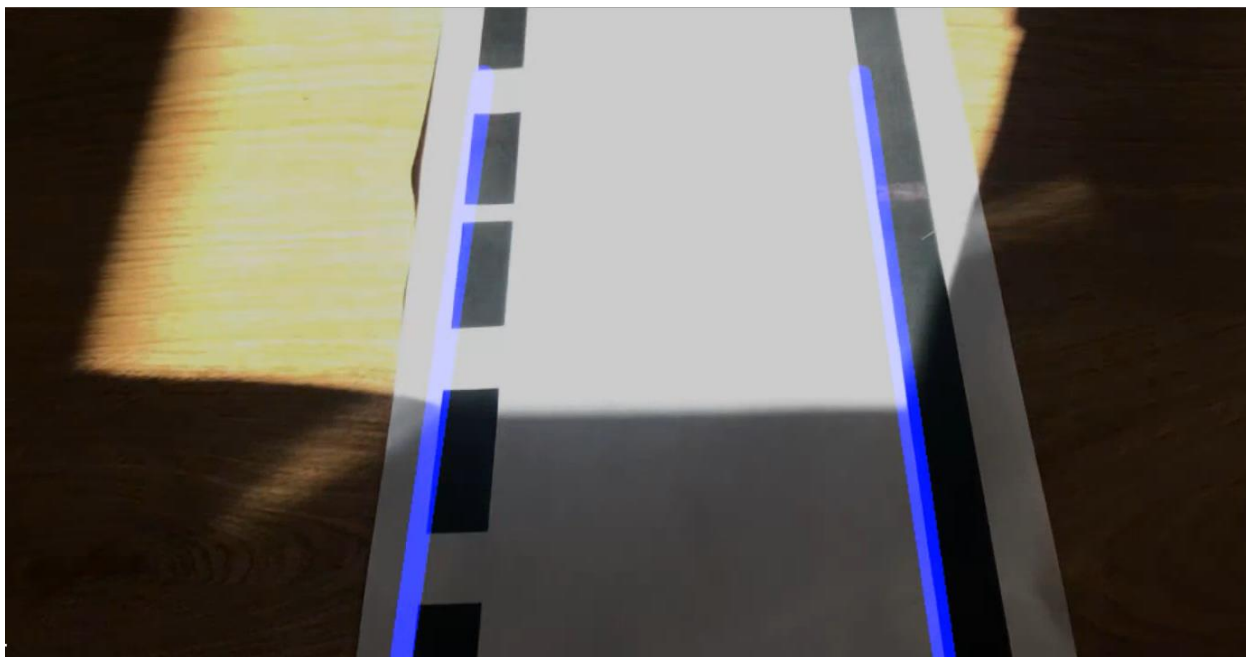


Рисунок 5.12. – Отображение результата при солнечном свете

Видно, что алгоритм хорошо справляется со своей задачей.

Заключение

В результате выполнения выпускной квалификационной работы была реализована программная система распознавания знаков дорожного движения, светофоров и дорожной разметки.

Для распознавания знаков дорожного движения и светофоров был выбран метод классификации объектов – каскад Хаара. А для распознавания дорожной разметки – метод, основанный на особенностях цвета и формы объекта распознавания.

Недостатком алгоритма Хафа является то, что он не умеет распознавать объекты, которые сильно отклоняются от ракурса того объекта, под который он обучался. Эту проблему можно решить, добавив в положительный набор данных больше фотографий объекта с самых разных ракурсов. Но в данном случае, при распознавании знаков и светофоров, это не является недостатком, так как камера (установленная в машине, либо на мобильном роботе) будет вести съемку под каким-то определенным углом.

Обнаружение прямых линий с помощью преобразования Хафа будет плохо работать на дороге с изогнутыми линиями. Если освещение на видеозаписи будет меняться очень быстро, то алгоритм также будет плохо справляться со своей задачей. Поэтому данный алгоритм в дальнейшем можно усовершенствовать так, чтобы он работал с кривыми линиями и при любой освещенности. Но лучших результатов, по моему мнению, можно достичь если подойти к решению данной задачи со стороны машинного обучения.

Результаты, полученные в работе, планируется внедрить в систему управления мобильного робота, моделирующего беспилотное транспортное средство.

Список литературы

1. Adrian Kaehler, Gary Bradski. Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library
2. Li Y., Chen L., Huang H., Li X., Xu W., Zheng L., Huang J. Nighttime lane markings recognition based on Canny detection and Hough transform // IEEE International Conference on Real-Time Computing and Robotics (RCAR). 2016. doi:10.1109/rcar.2016.7784064
3. OpenCV-Python Tutorials [электронный ресурс]. URL: <https://opencv-python-tutroals.readthedocs.io/en/latest/>
4. RoboCraft [электронный ресурс]. URL: <http://robocraft.ru/blog/computervision/>
5. Википедия [электронный ресурс]. URL: <https://www.wikipedia.org/>
6. Хабр [электронный ресурс]. URL: habr.com

Приложение 1

constants.py

```
# папка с положительными изображениями
GOOD_FOLDER = "good"

# папка с отрицательными изображениями
BAD_FOLDER = "bad"
# папка, в которую будут положены результаты обучения каскада Хаара
DATA_FOLDER = "data"

# файл, с описанием отрицательных изображений
BAD_TXT_FILE = "bad.txt"
# файл, с описанием положительных изображений
GOOD_TXT_FILE = "good.txt"
# файл с приведёнными к общему формату положительными изображениями
VEC_FILE = "vector.vec"

# Размер шаблона
GOOD_SIZE = (20, 20) # 25,45

# количество уровней каскада
NUM_STAGES = 14
# коэффициент, определяющий качество обучения
MIN_HIT_RATE = 0.999
# уровень ложной тревоги
MAX_FALSE_ALARM_RATE = 0.4
# общее количество имеющихся положительных примеров
NUM_OF_GOOD = 1000
# 80% от имеющихся положительных примеров
NUMGOOD = int(NUM_OF_GOOD * 0.80)
# общее количество имеющихся отрицательных примеров
NUMBAD = 3200
# использовать полный комплект Хаар-признаков
MODE = 'ALL'
# выделяемая под процесс память
MEMORY_SIZE = 1024
```

					ВКР-НГТУ-09.03.01-(16-В-1)-017-2020 (ПЗ)	Лист
						53
Изм.	Лист	№ докум.	Подп.	Дата		

get_img_from_video.py

```
import cv2
import os
from constants import GOOD_FOLDER

# Если нужно получить изображения из заранее снятого видео,
# то параметр GET_IMG_FROM_VIDEO должен быть True.
# Если нужно получить изображения из видеопотока в режиме реального
времени,
# то параметр GET_IMG_FROM_VIDEO должен быть False.
GET_IMG_FROM_VIDEO = True
# Видеофайл, из которого будут извлекаться кадры
video_file = 'C:\\Users\\Natalia\\Desktop\\video_signs\\IMG_3048.mp4'

# Получение изображений из видеопотока в режиме реально времени
def get_img_real_time():
    cap = cv2.VideoCapture(1)
    # Проверка на успешное открытие камеры
    if (cap.isOpened() == False):
        print("Error opening video file")
    # Читайте, пока видео не закончено
    i = 0
    while (cap.isOpened()):
        ret, frame = cap.read()
        i += 1
        if ret == True:
            cv2.imwrite(GOOD_FOLDER + '\\' + str(i) + '.jpg', frame)
            # Показать полученный кадр
            cv2.imshow('Frame', frame)
            # Пока не нажата клавиша Q на клавиатуре, видео будет продолжать
            воспроизводиться
            if cv2.waitKey(25) & 0xFF == ord('q'):
                break
        else:
            break
    cap.release()
    cv2.destroyAllWindows()
```

					ВКР-НГТУ-09.03.01-(16-В-1)-017-2020 (ПЗ)	Лист
						54
Изм.	Лист	№ докум.	Подп.	Дата		

```

# Получение изображений из заранее снятого видео
def get_img_from_video_file():
    # Создание объекта VideoCapture и чтение из входного файла
    cap = cv2.VideoCapture(video_file)
    # Проверка на успешное открытие файла
    if (cap.isOpened() == False):
        print("Error opening video file")
    # Чтение каждого кадра, пока видео не закончено
    i = 0
    while (cap.isOpened()):
        # Захват по кадрам
        ret, frame = cap.read()
        i += 1
        if ret == True:
            cv2.imwrite(os.path.join(GOOD_FOLDER, str(i) + '.jpg'), frame)
            # Показать полученный кадр
            cv2.imshow('Frame', frame)
            # Пока не нажата клавиша Q на клавиатуре, видео будет продолжать
            # воспроизводиться
            if cv2.waitKey(25) & 0xFF == ord('q'):
                break
            else:
                break
    # Очистить все окна и освободить память
    cap.release()
    cv2.destroyAllWindows()

if __name__ == '__main__':
    # если файл существует и он не пустой
    if (os.path.exists(GOOD_FOLDER) == True) and
    (len(os.listdir(GOOD_FOLDER)) != 0):
        # удалить содержимое папки вместе с самой папкой
        command = 'RMDIR /s /q ' + GOOD_FOLDER
        os.system(command)
        # создать новую пустую папку
        os.mkdir(GOOD_FOLDER)
        command2 = 'cd ' + GOOD_FOLDER
        os.system(command2)
    if GET_IMG_FROM_VIDEO:
        get_img_from_video_file()

```

```

else:
    get_img_real_time()
elif os.path.exists(GOOD_FOLDER) == False:
    os.mkdir(GOOD_FOLDER)
    if GET_IMG_FROM_VIDEO:
        get_img_from_video_file()
    else:
        get_img_real_time()
else:
    if GET_IMG_FROM_VIDEO:
        get_img_from_video_file()
    else:
        get_img_real_time()

```

get_positive_img.py

```

import cv2
import os
from constants import GOOD_FOLDER

# список опорных точек
refPt = []
# указание выполняется ли обрезка или нет
cropping = False

def click_and_crop(event, x, y):
    # глобальные переменные
    global refPt, cropping
    # если была нажата левая кнопка мыши, то начальные
    # координаты (x, y) записываются и указывается, что выполняется обрезка
    if event == cv2.EVENT_LBUTTONDOWN:
        refPt = [(x, y)]
        cropping = True
    # проверка на то, что левая кнопка мыши была отпущена
    elif event == cv2.EVENT_LBUTTONUP:
        # запись конечных координат (x, y) и указание того, что обрезка
        # завершена
        refPt.append((x, y))
        cropping = False

```

```

# рисование прямоугольника вокруг выделенной области
cv2.rectangle(image, refPt[0], refPt[1], (0, 255, 0), 2)
cv2.imshow("image", image)

def crop(image, file):
    # цикл продолжается пока не нажата клавиша
    while True:
        # вывод изображения на экран и ожидание нажатия клавиши
        cv2.imshow("image", image)
        key = cv2.waitKey(1) & 0xFF
        # если нажата клавиша "r", то выделенная область обрезки сбрасывается
        if key == ord("r"):
            image = clone.copy()
        # при нажатии клавиши "c" происходит выход из цикла
        elif key == ord("c"):
            break
        # если есть 2 опорные точки, то выделенная область обрезается
        # и выводится на экран
        if len(refPt) == 2:
            roi = clone[refPt[0][1]:refPt[1][1], refPt[0][0]:refPt[1][0]]
            cv2.imwrite(os.path.join(GOOD_FOLDER, file), roi)
            cv2.waitKey(0)
            cv2.destroyAllWindows()

if __name__ == '__main__':
    for root, dirs, files in os.walk(GOOD_FOLDER):
        for file in files:
            # чтение каждого изображения в указанной папке
            file_path = os.path.join(GOOD_FOLDER, file)
            image = cv2.imread(file_path)
            # изменение размера изображения для того, чтобы оно убралось на
            весь экран
            final_wide = 300
            r = float(final_wide) / image.shape[1]
            dim = (final_wide, int(image.shape[0] * r))
            image = cv2.resize(image, dim)
            clone = image.copy()
            cv2.namedWindow("image")
            # функция вызова мыши
            cv2.setMouseCallback("image", click_and_crop)
            crop(image, file)

```

haar_cascade_train.py

```
import cv2
import os
import subprocess

# для обучения каскада Хаара нужно раскомментировать одну из трех строчек
# например, если мы хотим обучить каскад для дорожной разметки, то нужно
# раскомментировать строку from constants_lines import *
from constants import *    # константы

# изменяем размер отрицательных изображений
def resize_bad_images():
    for img in os.listdir(BAD_FOLDER):
        _, file_extension = os.path.splitext(img)
        file_extension = file_extension.lower()
        if file_extension.endswith(".jpg") or file_extension.endswith(".png"):
            try:
                img_read = cv2.imread("%s/%s" % (BAD_FOLDER, img))
                resized_image = cv2.resize(img_read, (100, 100))
                cv2.imwrite("%s/%s" % (BAD_FOLDER, img), resized_image)

            except Exception as e:
                print(str(e))

def create_file_if_its_nul(file):
    if os.path.isfile(file) == 1 and os.stat(file).st_size != 0:
        command = 'TYPE nul > ' + file
        # очистить содержимое файла
        os.system(command)

# создание текстового файла с описанием отрицательных изображений
def create_bad_txt():
    for root, dirs, files in os.walk(BAD_FOLDER):
        for file in files:
            file_path = os.path.join(BAD_FOLDER, file)
            # создать файл с описанием отрицательных изображений
            my_file = open(BAD_TXT_FILE, "a")
            my_file.write(file_path + '\n')
            my_file.close()
```

					ВКР-НГТУ-09.03.01-(16-В-1)-017-2020 (ПЗ)	Лист
						58
Изм.	Лист	№ докум.	Подп.	Дата		

```

# создание текстового файла с описанием положительных изображений
def create_good_txt():
    for root, dirs, files in os.walk(GOOD_FOLDER):
        for file in files:
            file_path = os.path.join(GOOD_FOLDER, file)
            img = cv2.imread(file_path)
            img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            cv2.imwrite(os.path.join(GOOD_FOLDER, file), img_gray)
            image_shape = img.shape
            # создать файл с описанием отрицательных изображений
            my_file = open(GOOD_TXT_FILE, "a")
            my_file.write(file_path + ' 1 0 0 ' + str(image_shape[1]) + ' ' +
str(image_shape[0]) + '\n')
            my_file.close()

# создание пачки приведённых положительных изображений
def get_vector_file():

subprocess.check_call(["D:\\opencv\\build\\x64\\vc12\\bin\\opencv_createsamples"
,
                        "-info", GOOD_TXT_FILE,
                        "-vec", VEC_FILE,
                        "-w", str(GOOD_SIZE[0]),
                        "-h", str(GOOD_SIZE[1])])

# создаем итоговый каскад
def train_haar_cascade():
    if not os.path.exists(DATA_FOLDER):
        os.makedirs(DATA_FOLDER)

subprocess.check_call(["D:\\opencv\\build\\x64\\vc12\\bin\\opencv_traincascade",
                        "-data", DATA_FOLDER,
                        "-vec", VEC_FILE,
                        "-bg", BAD_TXT_FILE,
                        "-numStages", str(NUM_STAGES),
                        "-minHitRate", str(MIN_HIT_RATE),
                        "-maxFalseAlarmRate", str(MAX_FALSE_ALARM_RATE),
                        "-numPos", str(NUMGOOD),
                        "-numNeg", str(NUMBAD),

```

					БКР-НГТУ-09.03.01-(16-В-1)-017-2020 (ПЗ)	Лист
						59
Изм.	Лист	№ докум.	Подп.	Дата		

```

-w", str(GOOD_SIZE[0]),
-h", str(GOOD_SIZE[1]),
-mode", MODE,
-precValBufSize", str(MEMORY_SIZE),
-precIdxBufSize", str(MEMORY_SIZE)])

```

```

if __name__ == '__main__':
    resize_bad_images()
    create_file_if_its_nul(BAD_TXT_FILE)
    create_bad_txt()
    create_file_if_its_nul(GOOD_TXT_FILE)
    create_good_txt()
    create_file_if_its_nul(VEC_FILE)
    get_vector_file()
    train_haar_cascade()

```

detect_sign.py

```
import cv2
```

```

def crop_frame(image):
    if (len(image.shape) == 3):
        height, length, _ = image.shape
    else:
        height, length = image.shape
    return image[0: height // 3 * 2, 0: length]

```

```

if __name__ == '__main__':
    # путь к видеозаписи
    filepath = 'C:\\Users\\Natalia\\Desktop\\video_signs\\IMG_3146.mp4'
    # загрузка обученных классификаторов
    sign1_cascade =
cv2.CascadeClassifier('D:\\PythonProjects\\DETECT_TRAFFIC_SIGN\\Haar_cas
cad\\signs\\kirpich\\1\\data\\kirpich.xml')
    sign2_cascade =
cv2.CascadeClassifier('D:\\PythonProjects\\DETECT_TRAFFIC_SIGN\\Haar_cas
cad\\signs\\nalevo\\1\\data\\nalevo.xml')
    sign3_cascade =

```

					БКР-НГТУ-09.03.01-(16-В-1)-017-2020 (ПЗ)	Лист
						60
Изм.	Лист	№ докум.	Подп.	Дата		

```

cv2.CascadeClassifier('D:\\PythonProjects\\DETECT_TRAFFIC_SIGN\\Haar_cas
cad\\signs\\napravo\\1\\data\\napravo.xml')
sign4_cascade =
cv2.CascadeClassifier('D:\\PythonProjects\\DETECT_TRAFFIC_SIGN\\Haar_cas
cad\\signs\\pryamo\\1\\data\\pryamo.xml')
sign5_cascade =
cv2.CascadeClassifier('D:\\PythonProjects\\DETECT_TRAFFIC_SIGN\\Haar_cas
cad\\signs\\pryamonalevo\\1\\data\\pryamonalevo.xml')
sign6_cascade =
cv2.CascadeClassifier('D:\\PythonProjects\\DETECT_TRAFFIC_SIGN\\Haar_cas
cad\\signs\\pryamonapravo\\1\\data\\pryamonapravo.xml')
light_green =
cv2.CascadeClassifier('D:\\PythonProjects\\DETECT_TRAFFIC_SIGN\\Haar_cas
cad\\light\\green\\1\\data\\green.xml')
light_red =
cv2.CascadeClassifier('D:\\PythonProjects\\DETECT_TRAFFIC_SIGN\\Haar_cas
cad\\light\\red\\1\\data\\red.xml')
# чтение видеозаписи
cap = cv2.VideoCapture(filepath)
# либо чтение видеопотока в реальном времени
#cap = cv2.VideoCapture(1)
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))

out = cv2.VideoWriter('outpy.avi', cv2.VideoWriter_fourcc('M', 'J', 'P', 'G'), 10,
(frame_width, frame_height))

while (cap.isOpened()):
    ret, frame = cap.read()
    if (ret):
        # обрезаем каждый кадр
        cropped_frame = crop_frame(frame)
        # переводим кадр в ч/б
        gray_filered = cv2.cvtColor(cropped_frame, cv2.COLOR_BGR2GRAY)
        #scaler = StandardScaler()
        #gray = scaler.fit_transform(gray_filered)
        # функция поиска знаков
        signs_kirpich = sign1_cascade.detectMultiScale(gray_filered,
scaleFactor=1.15, minNeighbors=15,
minSize=(200, 200))

```



```

signs_nalevo = sign2_cascade.detectMultiScale(gray_filered,
scaleFactor=1.15, minNeighbors=20,
minSize=(200, 200))
signs_napravo = sign3_cascade.detectMultiScale(gray_filered,
scaleFactor=1.15, minNeighbors=20,
minSize=(200, 200))
signs_pryamo = sign4_cascade.detectMultiScale(gray_filered,
scaleFactor=1.15, minNeighbors=20,
minSize=(200, 200))
signs_pryamonalevo = sign5_cascade.detectMultiScale(gray_filered,
scaleFactor=1.15, minNeighbors=20,
minSize=(200, 200))
signs_pryamonapravo = sign6_cascade.detectMultiScale(gray_filered,
scaleFactor=1.15, minNeighbors=20,
minSize=(200, 200))

# функция поиска светофоров
light_green = light_green.detectMultiScale(gray_filered,
scaleFactor=1.15, minNeighbors=15,
minSize=(300, 600))
light_red = light_red.detectMultiScale(gray_filered, scaleFactor=1.15,
minNeighbors=15,
minSize=(300, 600))

# рисование прямоугольника вокруг найденного объекта
for (x, y, w, h) in signs_kirpich:
    cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 255, 0), 3)
    cv2.putText(frame, "NO ENTRY", (x, y-10),
cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 255, 0), 5)
for (x, y, w, h) in signs_nalevo:
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 3)
    cv2.putText(frame, "LEFT", (x, y-10),
cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 0, 255), 5)
for (x, y, w, h) in signs_napravo:
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 3)
    cv2.putText(frame, "RIGHT", (x, y-10),
cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 255, 0), 5)
for (x, y, w, h) in signs_pryamo:
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 255), 3)
    cv2.putText(frame, "STRAIGHT", (x, y-10),
cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 255, 255), 5)
for (x, y, w, h) in signs_pryamonalevo:
    cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 255), 3)
    cv2.putText(frame, "STRAIGHT AND LEFT", (x, y-10),

```

```

cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 0, 255), 5)
    for (x, y, w, h) in signs_pryamonapravo:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 3)
        cv2.putText(frame, "STRAIGHT AND RIGHT", (x, y-10),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 5)
    for (x, y, w, h) in light_green:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 255, 0), 3)
        cv2.putText(frame, "GREEN", (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 255, 0), 5)
    for (x, y, w, h) in light_red:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 255, 0), 3)
        cv2.putText(frame, "RED", (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 0, 0), 5)
    out.write(frame)
    cv2.imshow('frame', frame)
    cv2.waitKey(1)
else:
    break
k = cv2.waitKey(25) & 0xFF
if k == 27:
    break

cap.release()
cv2.destroyAllWindows()

```

Приложение 2

constants_lines.py

```
import numpy as np

#####
# размер ядра для фильтра Гаусса. Данный параметр
# определяет ширину и высоту окна фильтрации
kernel_size = 5
#####

#####
# нижний и верхний пороги для детектора границ Кэнни
low_threshold = 50
high_threshold = 150
#####

#####
# аргументы для функции "Преобразование Хафа"

# аргументы rho и тета задают требуемую разрешающую способность
# для прямых (т.е. квантизацию аккумуляторной плоскости)
rho = 2 # rho измеряется в пикселях
theta = np.pi/180 # тета измеряется в радианах
threshold = 50 # порог, при котором алгоритм сообщает о найденной прямой
min_line_length = 100 # минимальная длина возвращаемых отрезков
max_line_gap = 100 # требуемый промежуток между отрезками
#####
```

detect_line.py

```
import cv2
from constants_lines import *
import numpy as np
```

					ВКР-НГТУ-09.03.01-(16-В-1)-017-2020 (ПЗ)	Лист
Изм	Лист	№ докум.	Подп.	Дата		64

```

def draw_lines(img, lines, color=(255, 0, 0), thickness=10):
    # изменяем размер массива линий, делая его двухмерным
    lines = lines.reshape(lines.shape[0], lines.shape[2])
    # lines - массив линий, линия состоит из двух точек (x1,y1) и (x2,y2)
    # меняем массив линий на массив отдельных точек
    lines = np.reshape(lines, (lines.shape[0] * 2, 2))
    # считаем, что если точка находится справа от середины изображения,
    # то она принадлежит к правой полосе
    right_lines = np.array(list(filter(lambda x: x[0] > (img.shape[1] / 2), lines)))
    # считаем, что если точка находится слева от середины изображения,
    # то она принадлежит к левой полосе
    left_lines = np.array(list(filter(lambda x: x[0] < (img.shape[1] / 2), lines)))
    if len(right_lines) != 0 and len(left_lines) != 0 and len(lines) != 0:
        # находим нижнюю точку правой полосы
        min_right_x, min_right_y = np.amin(right_lines, axis=0)
        # находим нижнюю точку левой полосы
        min_left_x, min_left_y = np.amin(left_lines, axis=0)

        # находим аппроксимирующую функцию для правой и левой полосы по
        # методу наименьших квадратов
        right_curve = np.poly1d(np.polyfit(right_lines[:, 1], right_lines[:, 0], 2))
        left_curve = np.poly1d(np.polyfit(left_lines[:, 1], left_lines[:, 0], 2))

        # Решаем полином, вычисляем максимальное и минимальное значения
        # по x
        max_right_x = int(right_curve(img.shape[0]))
        min_right_x = int(right_curve(min_right_y))

        # Решаем полином, вычисляем значения x
        min_left_x = int(left_curve(img.shape[0]))
        max_left_x = int(left_curve(min_left_y))

        # находим самую низкую точку, для того чтобы полосы
        # начинались на одном уровне. А значение самой высокой точки
        # будет равно значению высоты изображения, для того чтобы линии
        # были отображены до конца изображения
        min_y = min(min_right_y, min_left_y)

        r1 = (min_right_x, min_y)
        r2 = (max_right_x, img.shape[0])
        cv2.line(img, r1, r2, color, thickness)

```

```

l1 = (max_left_x, min_y)
l2 = (min_left_x, img.shape[0])
cv2.line(img, l1, l2, color, thickness)

```

```

print(r1, ' ', r2)
print(l1, ' ', l2)

```

```
def search_lines(img):
```

```
    # преобразование Хафа для поиска прямых линий (возвращает 2 конечные точки линий)
```

```
    lines = cv2.HoughLinesP(img, rho, theta, threshold, np.array([]),
minLineLength=min_line_length,
maxLineGap=max_line_gap)
```

```
    # создаем массив такого же размера как исходное изображение,
заполненный нулями
```

```
    line_img = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)
    return line_img, lines
```

```
def region_of_interest(img):
```

```
    # размер изображения, h-высота, w-ширина
```

```
    h = img.shape[0]
```

```
    w = img.shape[1]
```

```
    # определение вершин для области интереса
```

```
    # (левая верхняя точка исходного изображения имеет координаты (0;0),
```

```
    # правая нижняя точка исходного изображения имеет координаты (x;y)).
```

```
    print(w,h)
```

```
    # левая нижняя точка области интереса
```

```
    bottom_left = (200, h)
```

```
    # левая верхняя точка области интереса
```

```
    top_left = (300, 50)
```

```
    # правая верхняя точка области интереса
```

```
    top_right = (600, 50)
```

```
    # правая нижняя точка области интереса
```

```
    bottom_right = (650, h)
```

```

# создание массива, определяющего область интереса на изображении
vertices = np.array([[bottom_left,
                        top_left,
                        top_right,
                        bottom_right]],
                    dtype=np.int32)
# изображение для рисования (маска)
image_to_draw = np.zeros_like(img)
# определения цвета для закрашивания области интереса
# в зависимости от входного изображения
(одноканальное/многоканальное)
if len(img.shape) > 2:
    channel_count = img.shape[2] # количество каналов у изображения (3 или
4)
    color = (255,) * channel_count
else:
    color = 255
# закрашивание полигона, определенного вершинами цветом заливки
cv2.fillPoly(image_to_draw, vertices, color)
# наложение маски поверх оригинального изображения, сохраняя каждый
пиксель
# изображения, если соответствующее значение маски равно 1
image_mask = cv2.bitwise_and(img, image_to_draw)
return image_mask

def prepare_img(img):
    # перевод изображения в черно-белое пространство
    gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    # применение к изображению фильтра Гаусса
    blur_img = cv2.GaussianBlur(gray_img, (kernel_size, kernel_size), 0)
    # применение к изображению детектора границ Кэнни
    Canny_img = cv2.Canny(np.uint8(blur_img), low_threshold, high_threshold)
    return Canny_img

if __name__ == "__main__":
    # путь к изображению
    path_img = 'D:\\PythonProjects\\detect_lines\\Lane-Detection-master\\Lane-
Detection-master\\test_images\\road6.jpg'

```

```

# чтение изображения
image = cv2.imread(path_img)
# проверка существует ли изображение
if image is None:
    print('Изображение отсутствует. Проверьте путь к изображению.')
final_wide = 960
r = float(final_wide) / image.shape[1]
dim = (final_wide, int(image.shape[0] * r))

# уменьшаем изображение до подготовленных размеров
resized = cv2.resize(image, dim, interpolation=cv2.INTER_AREA)
# предобработка изображения
Canny_img = prepare_img(resized)
# выделение области интереса
img_mask = region_of_interest(Canny_img)
cv2.imshow('can', img_mask)
cv2.waitKey(0)
# поиск линий на изображении
line_img, lines = search_lines(img_mask)
# рисование линий на изображении
draw_lines(line_img, lines, thickness=10)
lines_edges = cv2.addWeighted(resized, 0.8, line_img, 1, 0)
cv2.imshow('lines', lines_edges)
cv2.waitKey(0)
cv2.imwrite('lines_output.jpg', lines_edges)

```

detect_line_from_viseo.py

```

import cv2
from detect_line import draw_lines, search_lines, region_of_interest, prepare_img

video_file = 'C:\\Users\\Natalia\\Desktop\\video_signs\\IMG_3163.mp4'
# Создание объекта VideoCapture и чтение из входного файла

cap = cv2.VideoCapture(video_file)
#cap = cv2.VideoCapture(1)
# Проверка на успешное открытие файла
if (cap.isOpened() == False):
    print("Error opening video file")
# Чтение каждого кадра, пока видео не закончено
i = 0

```

```

frame_width = int(cap.get(3))
frame_height = int(cap.get(4))
out = cv2.VideoWriter('outpy.avi', cv2.VideoWriter_fourcc('M', 'J', 'P', 'G'), 10,
(frame_width, frame_height))
while (cap.isOpened()):
    # Захват по кадрам
    ret, frame = cap.read()
    i += 1
    if ret == True:
        final_wide = 960
        r = float(final_wide) / frame_width
        dim = (final_wide, int(frame_height * r))

        # уменьшаем изображение до подготовленных размеров
        resized = cv2.resize(frame, dim, interpolation=cv2.INTER_AREA)

        # предобработка изображения
        Canny_img = prepare_img(resized)
        # выделение области интереса
        img_mask = region_of_interest(Canny_img)
        # поиск линий на изображении
        line_img, lines = search_lines(img_mask)
        if lines is not None:
            # рисование линий на изображении
            draw_lines(line_img, lines, thickness=15)
            lines_edges = cv2.addWeighted(resized, 0.8, line_img, 1, 0)
            # Показать полученный кадр
            out.write(lines_edges)
            cv2.imshow('Frame', lines_edges)
        else:
            cv2.imshow('Frame', resized)

        # Пока не нажата клавиша Q на клавиатуре, видео будет продолжать
        воспроизводиться
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break
        else:
            break
# Очистить все окна и освободить память
cap.release()
cv2.destroyAllWindows()

```