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

Институт ИРИТ

Направление подготовки 09.03.01 Информатика и вычислительная техника
(код и наименование)

Направленность (профиль) образовательной программы Вычислительные машины, комплексы, системы и сети
(наименование)

Кафедра Вычислительные системы и технологии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

бакалавра
(бакалавра, магистра, специалиста)

Студента Смирнова Игоря Павловича группы 15-В-1
(Ф.И.О.)

на тему Программная система распознавания дорожной разметки
(наименование темы работы)

СТУДЕНТ:

_____ Смирнов И.П.
(подпись) (ф. и. о.)

_____
(дата)

РУКОВОДИТЕЛЬ:

_____ Гай В.Е.
(подпись) (ф. и. о.)

_____
(дата)

РЕЦЕНЗЕНТ:

_____ _____
(подпись) (ф. и. о.)

_____
(дата)

ЗАВЕДУЮЩИЙ КАФЕДРОЙ

_____ Мякиньков А.В.
(подпись) (ф. и.о.)

_____
(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 _____

Оглавление

Введение

Возможность видеть и распознавать объекты – естественная и привычная возможность для человека. Однако для компьютера, пока что – это чрезвычайно сложная задача. Сейчас предпринимаются попытки научить компьютер хотя бы малой части навыков, которые человек использует каждый день, даже не замечая этого.

Одно из самых первых изобретений, отчасти связанное, с компьютерным зрением это штрих-коды. Первый товар со штрих-кодом был продан в 1973 году. Это изобретение можно и не считать компьютерным зрением, так как для считывания штрих-кода используется не камера, а устройство с лазерным лучом, засвечивающим линии, сенсорами, которые считывают отражённое изображение, и микросхемой, переводящей полученную информацию в код, который имеет какой-либо смысл. Однако, нет точного описания, что можно считать компьютерным зрением, а что нет, а данные, так или иначе, получены с изображения, подаваемого на вход устройству, распознаны и полученный результат может быть использован.

Хотя и нет точного определения компьютерному зрению, всё же отрасль разделяется на несколько его видов по цели применения:

- Распознавание – классическая задача в компьютерном зрении, это определение содержат ли видеоданные некоторый характерный объект, особенность или активность. Эта задача может быть достоверно и легко решена человеком, но до сих пор не решена удовлетворительно в компьютерном зрении в общем случае: случайные объекты в случайных ситуациях, все остальные задачи так или иначе основываются на распознавании;
- Распознавание текста – важная задача имеющая применение в области оцифровывания текстов, написанных ещё до появления ЭВМ, в том числе распознавание текста, написанного «от руки», также, может применятся для мгновенного машинного перевода с иностранного языка по фотографии, отличается от классической задачи распознавания тем, что данные заранее подготовлены, то есть известно, что изображение содержит текст;
- Идентификация – задача распознания индивидуального экземпляра объекта, принадлежащего к какому-либо классу, к примеру: идентификация определённого человеческого лица или отпечатка пальцев;
- Обнаружение – видеоданные проверяются на наличие определенного условия, к примеру, обнаружение движения на охраняемой территории;

- Оценка движения – в общем случае это нахождение движущегося объекта и оценка, относительно некоторых других объектов на изображении, характера движения объекта: скорость, направление, траектория;
- Восстановление изображения – задача восстановления изображений - это удаление шума. Наиболее простым подходом к решению этой задачи являются различные типы фильтров, таких как фильтры нижних или средних частот. Более высокий уровень удаления шумов достигается в ходе первоначального анализа видеоданных на наличие различных структур, таких как линии или границы, а затем управления процессом фильтрации на основе этих данных.

Человечество стремится к автоматизации всех процессов, которые были созданы за большое количество времени. Уже несколько лет мировые автогиганты и IT-компании совместно разрабатывают беспилотные автомобили в основе которых лежит огромное количество инновационных разработок. Это направление развивается с большой скоростью и его актуальность сложно преуменьшить. Компьютерное зрение в такой системе одна из самых важных частей, т.к. именно с помощью зрения мы получаем основную информацию об окружающем нас пространстве и объектах. Так и для автомобиля компьютерное зрение должно стать обязательной частью, которая будет давать ему информацию о дорожной ситуации и задача об определении дорожной разметки это важная часть системы по определению дорожной ситуации.

1. Техническое задание

1.1 Назначение разработки и область применения

Разрабатываемая программная система предназначена для распознавания линий дорожной разметки на изображении, видеозаписи или видеопотоке.

Область применения разрабатываемой системы:

- интеграция с другими программными системами, к примеру, распознающими дорожные знаки, обнаруживающими объекты на дороге (другие автомобили, люди, животные и т.д.) для создания встраиваемого программного комплекса.

Разрабатываемая система может быть встроена в видеорегистраторы или бортовые компьютеры автомобиля для уведомления водителя о съезде с полосы движения или удержании автомобиля в полосе движения. В совокупности с другими средствами, также может уведомлять водителя о скоростных ограничениях или других важных ограничениях на участке дороги.

1.2 Технические требования

Рассмотрим требования, предъявляемые разрабатываемой системой к ЭВМ:

1. операционная система, наличие графического интерфейса не требуется;
2. требования к аппаратному обеспечению определяются операционной системой;
3. камера с подключением к системе;
4. дисплей;

Рассмотрим функциональность, которой должна обладать разрабатываемая система распознавания линий дорожной разметки:

1. перед началом работы система должна подключиться к видеокамере или загрузить видеозапись, для обработки данных;
2. система должна подготовить данные и провести анализ данных;
3. по совокупности признаков, система должна определить линии дорожной разметки на изображении;
4. система должна отобразить распознанные линии поверх исходного кадра видео и вывести изображение на экране или записать кадр в конец видеофайла.

2. Анализ технического задания

2.1 Выбор операционной системы

Выбор операционной системы определяет множество факторов работы системы и помимо разработки алгоритма, выбора языка программирования для реализации, необходимо выбрать ОС, для которой будет разрабатываться система. Рассмотрим подробнее варианты операционных систем применимо к разрабатываемой системе.

Windows – продукт компании Microsoft Corporation, занимает первое место на рынке операционных систем для ПК. Компания разрабатывает свои Windows-подобные операционные системы для смартфонов и планшетных компьютеров. Система имеет понятный большинству пользователей интерфейс и не требует серьёзной настройки после установки для рядового пользователя. Система может быть использована как основная ОС для разработки системы и её тестирования на видеозаписях с камеры, без необходимости ездить по дорогам с видеорегистратором для отладки кода.

Linux – это обобщённое название Unix-подобных операционных систем на базе ядра Linux. Операционные системы этой группы разрабатываются энтузиастами и компаниями со всего мира, так как ядро Linux имеет открытый исходный код. Распространяется в основном бесплатно в соответствии с моделью разработки свободного и открытого программного обеспечения. Первое появление Linux произошло в 1991 году, через 6 лет после Windows от Microsoft, что по меркам IT-индустрии довольно большой отрыв. Самые известные представители это Fedora, Ubuntu, Mint, Debian, Red Hat EL.

В большинстве случаев, операционные системы, установленные в видеорегистраторы, являются «кастомными» - разрабатываются отдельно компанией изготовителем для своей линейки регистраторов, однако некоторые из них основываются на дистрибутивах Linux, так как они бесплатны, имеют небольшой размер в занимаемом дисковом пространстве, по сравнению с ОС других семейств, и нетребовательны к аппаратной составляющей устройства и имеют большое сообщество, что облегчает разработку программных систем.

Программная система должна поддерживать возможность работы на платформе Linux как возможной операционной системе, встраиваемой в видеорегистраторы.

MacOS (OS X) – продукт компании Apple, Unix-подобная операционная система известная также как Macintosh Operating System. Является первой операционной системой с графическим интерфейсом пользователя, что делает её передовой в индустрии, системы на базе OS X распространяются только для аппаратов производства Apple, что не делает её популярной ввиду их дороговизны на территории РФ, однако это не мешает системам под управлением OS X занимать верхние строчки самых используемых ОС в мире.

Разработка под операционную систему MacOS нецелесообразна ввиду отсутствия у компании каких-либо систем ориентированных на работу в качестве видеорегистраторов или им подобных устройств.

Android – Unix-подобная операционная система, разрабатываемая для смартфонов, планшетов, различных гаджетов, игровых приставок, телевизоров и т.д. Самую большую долю на рынке систем под управлением OS Android занимают смартфоны и планшеты, операционная система лидирует в сегменте бюджетных смартфонов и в целом занимает первое место среди ОС для гаджетов.

В 2015 году появилась поддержка систем, встраиваемых в автомобиль в том числе как бортовой компьютер, что делает эту платформу очень интересной для разработки в перспективе.

На личном опыте, к минусам разработки под операционную систему Android отнесу трудности с оптимизацией кода, сложность кроссплатформенной разработки приложений с качественным графическим интерфейсом, а также ограничением в выборе языка программирования: разработка под ОС Android целесообразна только на Java и Kotlin.

Исходя из поставленной задачи можно сделать вывод, что приложение должно иметь возможность запуска как минимум на Linux и Windows платформах, а в лучшем случае не иметь зависимости от ОС, на которой оно запускается.

2.2 Выбор языка программирования

Выбор языка программирования немаловажная часть требований к конечному продукту. Каждый язык программирования обладает рядом свойств, которые отличают его от других, и конечном результате, неправильный выбор языка программирования может привести к большим сложностям в решении таких задач, которые в другом языке программирования считаются элементарными и не требуют большого количества времени на их решение.

Рассмотрим некоторые известные языки программирования, с учётом уже известного требования к кроссплатформенности разрабатываемой системы.

C++ – язык программирования, разработанный в 1983 году Бьерном Страуструпом. Компилируемый, статически типизированный язык программирования, поддерживающий основные парадигмы программирования, такие как Объектно-ориентированное программирование, процедурное программирование, обобщённое программирование. Имеет поддержку многопоточного программирования и при грамотной разработке ПО является самым быстродейственным и эффективным языком. Имеет применение в любой сфере программирования, от написания пользовательских приложений, до разработки операционных систем, компиляторов и драйверов. Однако, разработка кроссплатформенных приложений на C++ невозможна, так как написания ПО связана с особенностями разработки под конкретную операционную систему.

Java – объектно-ориентированный язык программирования со строгой типизацией, разработанный компанией Sun Microsystems в 1995 году. Является одним из самых популярных языков программирования для разработки прикладных приложений. В отличии от C++ имеет возможность кроссплатформенного запуска за счёт виртуальной Java-машины. Код написанный на языке Java исполняется не операционной системой, а виртуальной машиной, которая уже в свою очередь установлена на операционную систему и имеет необходимые зависимости. Таким образом код может быть написан один раз и запускаться на разных платформах. Большим минусом этого языка является его скорость работы, которая гораздо меньше чем у языков программирования, не использующих виртуальные машины для исполнения своего кода.

JavaScript – самый популярный по результатам 2018 года язык программирования, является реализацией ECMAScript. Это интерпретируемый язык программирования разрабатывался с целью сделать язык программирования похожим на Java, но проще для

использования непрограммистами, в следствии чего он, к примеру, имеет нестрогую типизацию.

Своей популярностью JavaScript обязан развитию сайтостроения, где нашёл большое применение у людей, не имеющих опыта в программировании, но желающих улучшить свой сайт и сделать его отличным от остальных в сети. Язык имеет огромное сообщество, но по большей части оно состоит из Web-разработчиков, что характеризует этот язык как язык программирования для сайтов и их серверных частей.

Python – высокоуровневый, интерпретируемый язык программирования разработанный в 1991 году Гвидо ван Россумом. Python ориентирован на увеличение читаемости кода и упрощения разработки с его использованием, в следствии чего активно используется в создании прототипов различных проектов, от которых не требуется оптимальная по времени и памяти работа. Однако это не единственная возможность его применения, Python имеет большое сообщество и огромное количество модулей, написанных разработчиками по всему миру, что сводит решение типичных задач с его помощью к поиску необходимого модуля и его реализации в своих задачах. Большое распространение получил в сфере автоматизации процессов, аналитике данных и создании математических моделей. Python основывается на виртуальной машине для запуска своего кода и как следствие программы разработанные на Python являются кроссплатформенным.

Изучив плюсы и минусы популярных языков программирования, нельзя однозначно решить какому языку программирования отдать предпочтение, главным параметром является кроссплатформенность, которому удовлетворяет Java и Python. Однако учитывая, что оба языка имеют относительно одинаковую скорость работы, предпочтение отдадим языку программирования Python, так как работа с компьютерным зрением предполагает обработку большого количества данных, а также язык даёт большее количество инструментов для разработки систем в данной области.

2.3 Выбор среды разработки

Исходя из выбранного языка программирования, посмотрим на самые популярные среды разработки, используемые в языке Python.

PyCharm – самой популярной интегрированной средой разработки для языка Python является PyCharm. Разработка компании Jetbrains, пользуется большой популярностью среди разработчиков из-за того, что разработана именно для языка Python и имеет большое количество удобных функций, которые ускоряют работу программистов. Однако, распространяется данная среда разработки коммерческой основе. Имеется специальная версия для студентов, которая не включает в себя все возможности коммерческой версии, однако для разработки её достаточно.

Eclipse – свободно распространяемая среда разработки, поддерживает множество языков программирования за счёт модульности. Необходимый модуль можно скачать из сети Интернет и установить в Eclipse. Данная среда разработки не имеет большого количества функций, так как разрабатывается сообществом и улучшения в ней появляются не часто и не много. Удобство данной среды в её размере, она не занимает много места на системе и не требовательна к аппаратной составляющей, чего нельзя сказать про PyCharm.

Atom и Sublime text – продвинутые текстовые редакторы с возможностью подсветки кода, множественной автозамены, удобному поиску по файлам и структуре проекта, однако не являются полноценными средами разработки. Помимо установленной программы, необходимо иметь открытой консоль, для запуска и отладки программ.

Jupyter – интерактивная среда разработки, устанавливаемая непосредственно в Python. По своей сути это веб сервер, запускаемый на системе, который поддерживает работу с несколькими проектами. Интерактивность системы заключается в её модульности, каждый блок кода, запущенный и отработавший в своём месте остаётся в памяти с результатом своего выполнения и нет необходимости его запускать интерпретацию всей программы ещё раз, для исполнения новых строчек кода, добавленных в конце проекта. Эта система пользуется большой популярностью среди аналитиков данных, математиков и физиков. Она позволяет выводить графики, не создавая новых окон, как необходимо делать с любыми другими средами разработки, а прямо следом за кодом, который выводит график, после чего продолжить расчёты или разработку опираясь на визуальные данные полученные ранее.

Microsoft Visual Studio - полнофункциональная IDE от Microsoft, которая во многом сопоставима с Eclipse. Доступная на Windows и Mac OS, Visual Studio представлена как в бесплатном (Community), так и в платном (Professional и Enterprise) вариантах. Visual Studio позволяет разрабатывать приложения для разных платформ и предоставляет свой собственный набор расширений. Python Tools for Visual Studio (PTVS) позволяет писать на Python в Visual Studio и включает в себя Intellisense для Python, отладку и другие инструменты.

Опираясь на условия задачи и имеющийся опыт в разработке с использованием различных языков программирования, остановимся на Jupyter в качестве среды разработки.

2.4 Обзор существующих систем по обнаружению линий дорожной разметки

Системы по распознаванию дорожной разметки создаются в передовых автокомпаниях совместно с IT компаниями. Все они основываются на компьютерном зрении, но имеют разный подход к реализации алгоритмов для распознавания. Основных способа для распознавания дорожной разметки два: основанный на анализе и основанный на обучении.

Обучаемый метод основывается на машинном обучении. Машинное обучение (Machine Learning) — обширный подраздел искусственного интеллекта, изучающий методы построения алгоритмов, способных обучаться. Подход довольно прост в реализации и в то же время очень сложен. Простота в том, что алгоритмы по машинному обучению сейчас широко распространены и написание своего станет не такой большой проблемой как обучение сети. Так как разметка данных это довольно трудоёмкий процесс, а обучение нейросети требует большого количества размеченных данных, этот подход не пользуется популярностью. Помимо разметки данных, машинное обучение требовательно к аппаратному обеспечению, на котором оно производит обучение. Для обучения на трёх тысячах изображений, необходимо иметь чип GPU последнего поколения и 3-4 часа времени, которое система будет производить обучение. Плюсом такой системы является большая точность, при правильном обучении системы.

Второй метод заключается в анализе изображения по таким критериям как цвет, угол и наклон линий, то есть на анализе текущего изображения, не основываясь на данных полученных ранее. Такой метод имеет большее число возможных ошибок на этапе работы, однако для работы ему не требуется мощное аппаратное обеспечение и время на обучение системы. С другой стороны, данный метод имеет большое количество реализаций, что говорит о его популярности и доступности.

2.5 Выбор подхода к решению задачи по обнаружению линий дорожной разметки

В разрабатываемой системе будем использовать метод основанный на анализе изображения для решения задачи, так как он более разнообразный в возможности решении поставленной задачи.

При решении задачи подобного типа необходимо иметь библиотеку поддерживающую работу с компьютерным зрением, на языке Python. Компания Intel разрабатывает библиотеку для компьютерного зрения OpenCV. Она применяется в некоторых продуктах самой компании и улучшается за счёт различных оптимизаций на низком уровне на процессорах компании. Так как библиотека распространяется бесплатно, будем использовать её в своей системе.

Алгоритм распознавания дорожной разметки можно разделить на три этапа (Рисунок 1):

- Первый этап заключается в предварительной обработке кадра и его векторизации.
- Второй этап работы алгоритма - обновление информации о положении полосы дорожной разметки.
- Третий этап – наложение выбранных линий на исходный кадр и вывод их на экран.

В результате наложения мы получим кадр с наложенными двумя линиями, поверх линии разметки.

Рисунок 1. – Алгоритм распознавания линий дорожной разметки

3. Разработка структуры системы поиска линий дорожной разметки

3.1 Разработка общей структуры системы

По своей сути алгоритм распознавания дорожной разметки — это линейный алгоритм с поочерёдным применением различных масок, фильтров и каждое следующее изображение исследуется алгоритмом заново. Алгоритм работы системы представлен на рисунке 2:

Рисунок 2. – Структура системы

На вход алгоритму подаётся видеопоток с камеры, находящейся в салоне автомобиля. Это может быть видеорегистратор, подключенный к компьютеру или специально установленная камера. Первым шагом система должна получить очередной кадр из видео, для дальнейшей обработки.

3.2 Разработка алгоритма предварительной обработки данных

Перед непосредственным определением линий дорожной разметки, необходимо подготовить изображение. Первым этапом является смена цветового пространства.

Чаще всего для хранения цифровых изображений используется цветовое пространство RGB. В нем каждой из трех осей (каналов) присваивается свой цвет: красный, зеленый и синий. На каждый канал выделяется по 8 бит информации, соответственно, интенсивность цвета на каждой оси может принимать значения в диапазоне от 0 до 255. Все цвета в цифровом пространстве RGB получаются путем смешивания трех основных цветов. К сожалению, RGB не всегда хорошо подходит для анализа информации. Эксперименты показывают, что геометрическая близость цветов достаточно далека от того, как человек воспринимает близость тех или иных цветов друг к другу.
Но существуют и другие цветовые пространства. Весьма интересно в нашем контексте пространство HSV (Hue, Saturation, Value). В нем присутствует ось Value, обозначающая количество света. На него выделен отдельный канал, в отличие от RGB, где это значение нужно вычислять каждый раз. Фактически, это черно-белая версия изображения, с которой уже можно работать. Hue представляется в виде угла и отвечает за основной тон. От значения Saturation (расстояние от центра к краю) зависит насыщенность цвета.

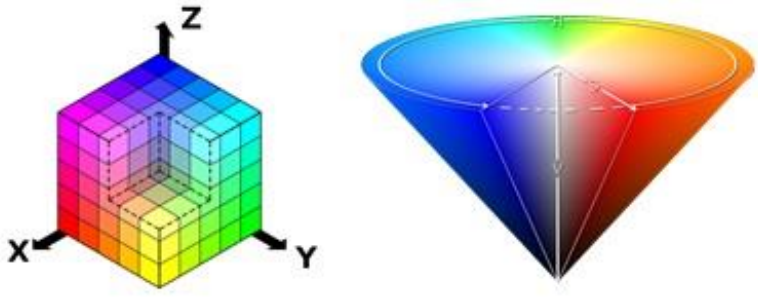

Рисунок 3. – Представление RGB пространства (слева) и HSV пространства (справа)

HSV гораздо ближе к тому, как мы представляем себе цвета. Если показать человеку в темноте красный и зеленый объект, он не сможет различить цвета. В HSV происходит то же самое. Чем ниже по оси V мы продвигаемся, тем меньше становится

разница между оттенками, так как снижается диапазон значений насыщенности. На схеме это выглядит как конус, на вершине которого предельно черная точка (рисунок 3).

Второй этап предварительной обработки изображения — это бинаризация, преобразование изображения в бинарную маску. Цвета для этой маски мы указываю самостоятельно. На выходе получим чёрное изображение с белыми пятнами на тех местах где цвет был в указанном диапазоне. Так как нам необходимы линии разметки, которые нанесены белой или жёлтой краской, указываем соответствующий диапазон. Факт того, что разметка не идеально белая покрывается тем, что мы перешли в другое цветовое пространство и реальный серый цвет разметки или тень на изображении будет восприниматься алгоритмом как белый со сниженной насыщенностью. Таким образом получим бинарную маску с объектами белого или жёлтого цвета.

3.3 Разработка алгоритма поиска прямых линий на изображении

Поиск всех линий на изображении относится к части его векторизации. В результате данного этапа необходимо иметь координаты всех линий на изображении для дальнейшей их обработки. Для этого необходимо последовательно применить два преобразования, которые лежат в основе компьютерного зрения: алгоритм Кэнни и алгоритм Хафа.

Алгоритм Кэнни (анг. Canny) – этот алгоритм нахождения границ объектов.

Первым шагом алгоритм устраняет шум в исходном изображении с помощью фильтра Гаусса. Функция Гаусса для двумерного случая:

$$G\_gen(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

где параметр σ – параметр степени сглаживания, чем он выше, тем больше сглаживается изображение.

Вторым шагом является нахождение краёв путём взятия градиента изображения. Оператор Собеля часто применяют для поиска границ. По своей сути это оператор, вычисляющий приближение градиента яркости изображения. Результатом его работы в каждой точке изображения является либо вектор градиента яркости в точке, либо его норма. Оператор Собеля основан на свёртке изображения небольшими целочисленными фильтрами в вертикальном и горизонтальном направлениях, поэтому его относительно легко вычислять. Оператор использует ядра 3x3, с которыми свёртывают исходное изображение для вычисления приближенных значений производных по горизонтали и по вертикали.

Третьим шагом алгоритма Canny является подавление не-максимумов для полученных границ, это даст тонкую линию в конечном изображении. Пример изображения, с применённым к нему алгоритмом Кэнни – рисунок 4.

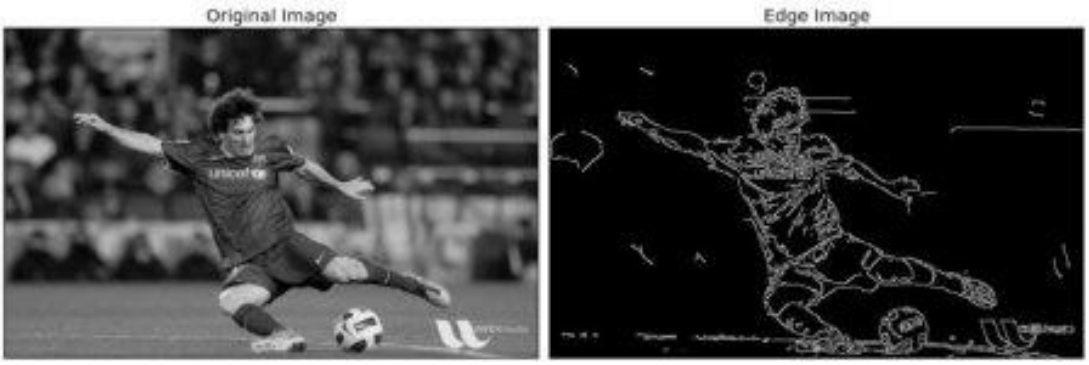

Рисунок 4. - Исходное изображение и результат работы алгоритма Canny

Теперь, имея изображение, на котором можно увидеть только очертания объектов, применим алгоритм Хафа (или преобразование Хафа) для поиска всех прямых линий на изображении - это метод для поиска простых форм на изображении с применением процедуры голосования.

Преобразование Хафа основывается на представлении искомого объекта в виде параметрического уравнения. Параметры этого уравнения представляют фазовое пространство (т.н. аккумуляторный массив/пространство, пространство Хафа).

Теперь, берётся двоичное изображение и перебираются все точки границ. Делается предположение, что точка принадлежит линии искомого объекта — таким образом, для каждой точки изображения рассчитывается нужное уравнение и получаются необходимые параметры, которые сохраняются в пространстве Хафа.

Финальным шагом является обход пространства Хафа и выбор максимальных значений, за которые «проголосовало» больше всего пикселей изображения, что и даёт нам параметры для уравнений искомого объекта.

В результате предварительной обработки изображения алгоритм имеет набор всех линий на изображении, которые не отсеялись двоичной маской.

3.4 Разработка алгоритма принятия решения

На данном этапе алгоритм должен принять на вход массив координат всех линий на изображении и решить какие являются частью дорожной разметки, а какие нет.

Для начала, необходимо определить признаки линии дорожной разметки. Они разделяются на два типа: постоянные и переменные.

К постоянному типу относятся следующие признаки:

- линия кандидат должна иметь некоторый небольшой наклон относительно вертикали, так как изображение с камеры находится в перспективе;
- линия кандидат должна быть ниже горизонта.

К признакам переменного типа отнесём следующие:

- линия кандидат должна быть достаточно близко в координатах кадра к тем, что были в нескольких кадрах ранее;
- уклон у линии кандидата не должен отличаться сильно от уклона линии уже определённой алгоритмом дорожной разметки в предыдущих кадрах;

Различие между признаками постоянного и переменного типа обусловлено тем, что признаки переменного типа сохраняются в памяти программы и постоянно обновляются на основании новых данных, обработанных алгоритмом, в то время как признаки постоянного типа не меняют своих характеристик на протяжении всего времени работы алгоритма.

На основе вышеупомянутых признаков, алгоритм принимает решение о принадлежности той или иной линии к линии дорожной разметки. Для принятия решения необходимо наличие у линии кандидата как минимум трёх признаков. При наличии нескольких кандидатов с тремя или четырьмя признаками выбирается тот, у которого признаков больше, т.е. четыре, если кандидатов с одинаковым количеством признаков несколько – кадр пропускается и алгоритм начинает обработку следующего кадра.

Необходимо принять во внимание, что алгоритм не знает положение дорожной разметки в самом начале своей работы и не имеет заполненного буфера предыдущих кадров, что исключает признаки переменного типа из рассмотрения на момент запуска алгоритма и на момент, когда дорожная разметка пропадает из поля зрения камеры, к примеру внезапное отсутствие дорожной разметки на дорожном полотне или приближение к впередиидущему автомобилю настолько близко, что дорожная разметка пропадает из

вида камеры. Такие ситуации являются для алгоритма нестандартными. При появлении дорожной разметки в поле зрения камеры, алгоритм начнёт свою работу с начала.

Таким образом, на момент наступления нестандартных ситуаций, алгоритм должен опираться только на признаки постоянного типа, не принимая во внимание отсутствие признаков переменного типа.

4. Разработка программных средств

Программная система распознавания линий дорожной разметки состоит из нескольких функций, каждая из которых в свою очередь определяет определённую часть алгоритма, а также несколько вспомогательных функций, которые улучшают работу программы, но не влияют непосредственно на алгоритм определения. Рассмотрим основные этапы алгоритма, реализованные в программном коде.

Применение бинарной маски, это первая стадия алгоритма по распознаванию дорожной разметки. Для того, чтобы алгоритм верно определил линии разметки необходимо облегчить ему задачу, убрав с изображения все цвета, кроме цветов линий разметки. Как известно это белый и жёлтый цвета. Для отсечения всех остальных цветов и применяется бинаризация изображения. Цвета в HSV пространстве задаются тремя переменными, h (hue): от 0 до 359 – цвет, s (saturation) и v (value): от 0 до 100 – насыщенность и яркость в процентах. Цвета для маски задаются переменные и могут быть удобно изменены при необходимости.

```
# White and yellow color thresholds for lines masking.
# Optional "kernel" key is used for additional morphology
WHITE_LINES = { 'low_th': to_opencv_hsv(0, 0, 80),
                'high_th': to_opencv_hsv(359, 10, 100) }

YELLOW_LINES = { 'low_th': to_opencv_hsv(49, 10, 100),
                 'high_th': to_opencv_hsv(57, 18, 100)}
```

Стоит принять во внимание, что библиотека OpenCV, которая используется для работы с компьютерным зрением, принимает входное видео не в RGB цветах, а BGR – то есть каналы красного и голубого цвета поменяны местами, что не всегда очевидно при разработке.

Функция get_lane_lines_mask принимает на вход два параметра, изображение, с которым необходимо работать и список ранее определённых цветов, бинарная маска которых будет вычисляться. Необходимо пояснить, что для каждого из определённых ранее цветов есть два параметра: low_th и high_th. В первый записывается нижнее значение цвета, то есть к примеру серый цвет, а во второй параметр записывается верхнее значение цвета – чистый белый цвет. Изображение до применения маски - Рисунок 5.

```
def get_lane_lines_mask(hsv_image, colors):
    """
    Image binarization using a list of colors. The result is a binary mask
    which is a sum of binary masks for each color.
    """
    masks = []
    for color in colors:
        if 'low_th' in color and 'high_th' in color:
            mask = cv2.inRange(hsv_image, color['low_th'], color['high_th'])
            if 'kernel' in color:
                mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, color['kernel'])
            masks.append(mask)
        else: raise Exception('High or low threshold values missing')
    if masks:
        return cv2.add(*masks)
```

Результатом работы функции является бинарное изображение - рисунок 6:

Рисунок 5. – Изображение до применения маски

Рисунок 6. – Бинарная маска изображения.

Следующим этапом является применение алгоритма Канни и OpenCV предоставляет удобную функцию для работы с данным преобразованием, так как это одно из самых необходимых преобразований в компьютерном зрении.

```
def canny(img, low_threshold, high_threshold):
    """Applies the Canny transform"""
    return cv2.Canny(img, low_threshold, high_threshold)
```

На вход можно подать любое изображение, однако следуя структуре алгоритма, на вход подаётся бинарная маска изображения (рисунок 6). В результате работы алгоритма на выходе имеется изображение со всеми линиями, отражающими переход от светлого участка изображения к тёмному (рисунок 7).

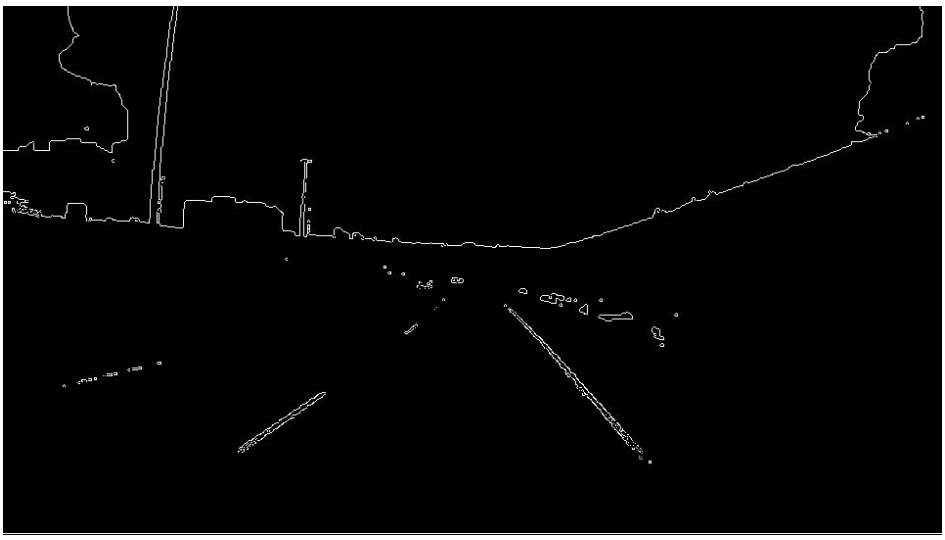

Рисунок 7. – Результат работы алгоритма Канни

Следующим этапом является применение алгоритма Хафа. Как известно, алгоритм находит все формы, определённые формулой. Существует отдельный метод поиска линий на изображении, так как поиск линий, это самое популярное применение данного алгоритма в компьютерном зрении.

```
def hough_line_transform(image, rho, theta, threshold, min_line_length, max_line_gap):
    """
    Returns a list of Line instances which are considered segments of a lane.
    """
    lines = cv2.HoughLinesP(image, rho, theta, threshold, np.array([]),
                            minLineLength=min_line_length, maxLineGap=max_line_gap)
    if lines is not None:
        filtered_lines = list(filter(lambda l: l.candidate, map(lambda line: Line(*line[0]), lines)))
        return filtered_lines
    else: return None
```

Результатом работы является массив линий кандидатов, которые необходимо проверить по определённым признакам. Применяя вспомогательную функцию, отобразим все линии, найденные алгоритмом Хафа на изображении (рисунок 8)

Рисунок 8. – результат работы алгоритма Хафа.

Определение линии кандидата, самая важная часть алгоритма. В результате работы преобразования Хафа, алгоритм может иметь большое множество линий, среди которых необходимо выбрать те, которые удовлетворяют условиям.

Функция candidate принимает на вход линию и последовательно проверяет её по условиям, описанным ранее.

```
def candidate(self):
    """
    1. The line cannot be horizontal and should have a reasonable slope.
    2. The difference between lane line's slope and this hough line's cannot be too high.
    3. The hough line should not be far from the lane line it belongs to.
    4. The hough line should be below the vanishing point.
    """
    if abs(self.a) < Lane.MOSTLY_HORIZONTAL_SLOPE: return False
    lane_line = getattr(Lane, self.lane_line)
    if lane_line:
        if abs(self.a - lane_line.coeffs[0]) > Lane.MAX_SLOPE_DIFFERENCE: return False
        if self.distance_to_lane_line > Lane.MAX_DISTANCE_FROM_LINE: return False
        if self.y2 < Lane.left_line.vanishing_point[1]: return False
    return True
```

В результате работы функции, линии поданной на вход присваивается значение, характеризующее подходит линия по условиям алгоритма или нет. После сбора информации о всех линиях, алгоритм выбирает те, у которых характеризующее значение максимально и отображает их на исходном изображении.

5. Тестирование системы

Тестирование систем, встраиваемых в транспортные средства проводят с особым контролем качества, покрывая самые незначительные изменения в функциональности большим количеством тестов. Однако, данный прототип не встраивается в автомобиль как система, от которой зависит безопасность людей, находящихся в транспортном средстве или рядом с ним.

Для тестирования системы было выбрано изображений из видео, взятых из сети интернет.

Тест №1, яркое изображение с чёткими белыми линиями, рисунки 9-11:

Рисунок 9. - Исходное изображение

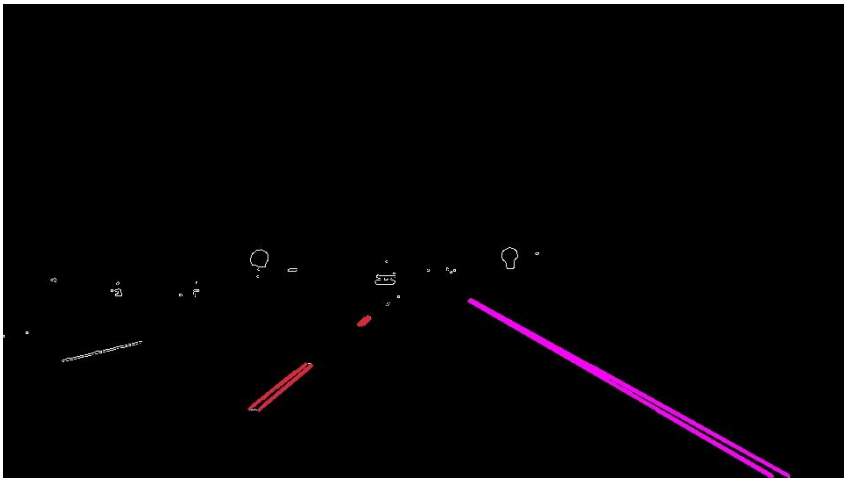

Рисунок 10. - Бинаризированное изображение с найденными линиями

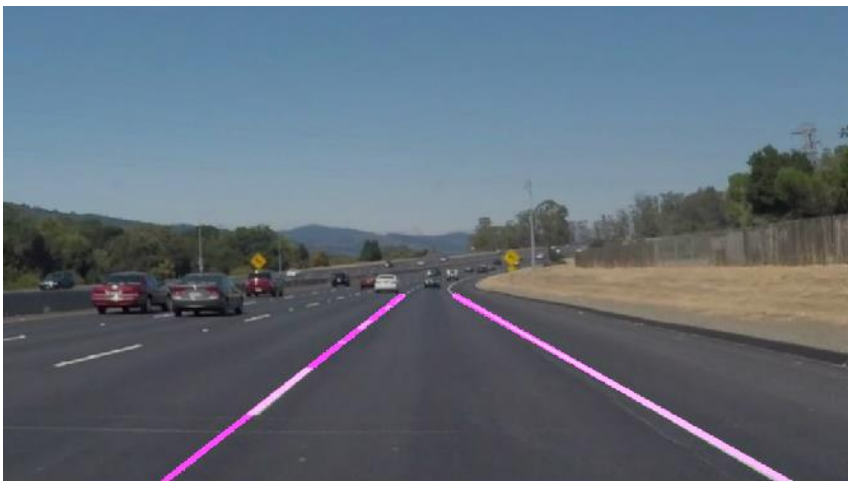

Рисунок 11. - Результат работы алгоритма

Тест №2, яркое изображение с чёткими белой и жёлтой линиями, рисунки 12-14:

Рисунок 12. - Исходное изображение

Рисунок 13. - Бинаризированное изображение с найденными линиями

Рисунок 14. - Результат работы алгоритма

Тест №3, яркое изображение с белым небом, с чёткими белыми линиями, рисунки 15-17:

Рисунок 15. - исходное изображение

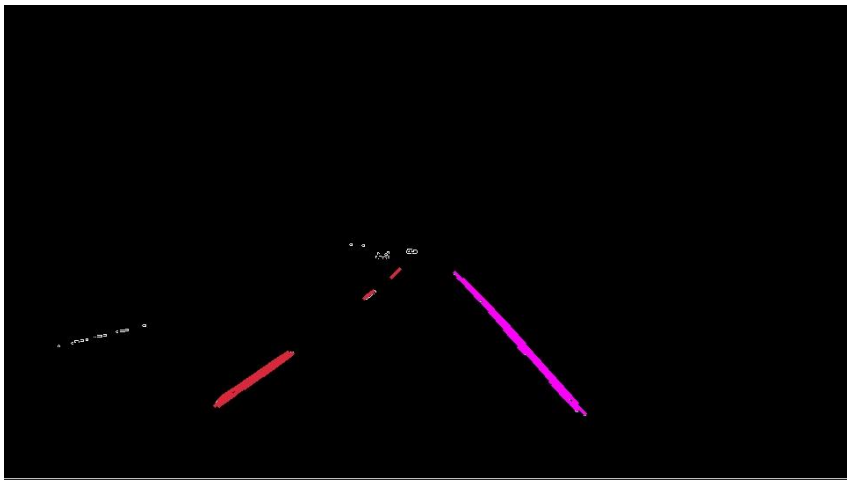

Рисунок 16. - бинаризированное изображение с найденными линиями

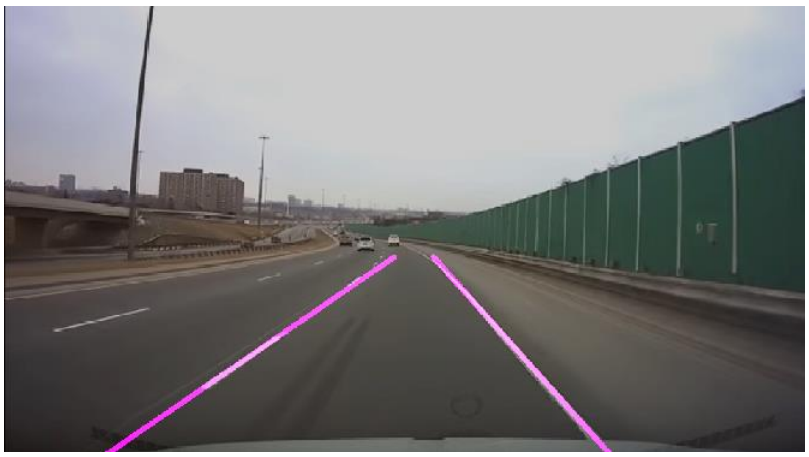

Рисунок 17. - Результат работы алгоритма

Тест №4: Прерывистые линии со сплошными линиями у края дороги, рисунки 18-20

Рисунок 18. - Исходное изображение

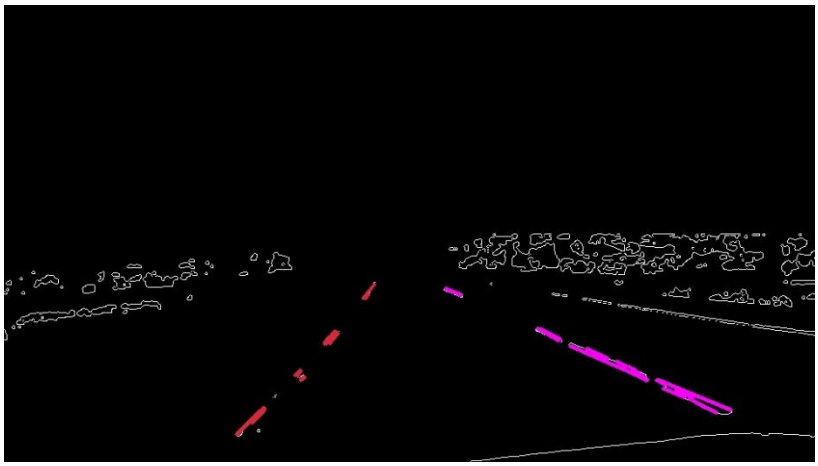

Рисунок 19. - Бинаризированное изображение с найденными линиями

Рисунок 20. - Результат работы алгоритма

Тест №5: Белые линии. Изображение преобладанием жёлтых тонов, рисунки 21-23

Рисунок 21. - Исходное изображение

Рисунок 22. - Бинаризированное изображение с найденными линиями

Рисунок 23. - Результат работы алгоритма

Результаты теста показывают, что при яркой дорожной разметке и ясной погоде, алгоритм справляется со своей задачей, 4 из 5 тестов пройдены.

Однако если при тех же значениях цветов жёлтого и белого цвета, алгоритму подать на вход изображения с пасмурной погодой или плохим освещением, то он теряется и не может найти разметку, это видно из бинарной маски, которая не показывает ничего (рисунок 24)

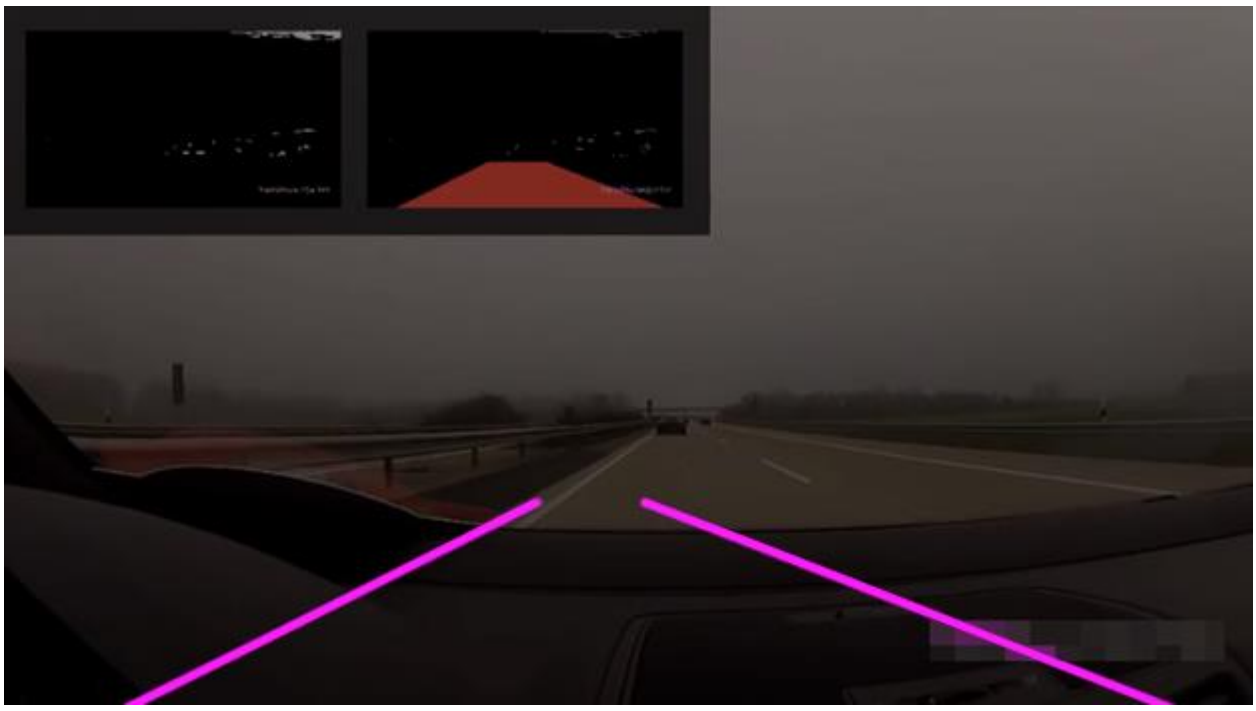

Рисунок 24. – Работа алгоритма с плохой освещённостью

Заключение

В результате выполнения выпускной квалификационной работы был написан прототип системы по распознаванию дорожной разметки.

Для распознавания линий дорожной разметки был выбран метод распознавания алгоритмический метод путём нахождения линий дорожной разметки на каждом отдельном кадре и отображения результата на исходном изображении. Большая часть работы была проделана по изучению открытой библиотеки OpenCV для работы с компьютерным зрением.

Созданная программная система предназначена для распознавания линий дорожной разметки на видео или изображениях, подаваемых системе. Система хорошо справляется с изображениями обладающими определёнными характеристиками, однако не с любыми изображениями, что говорит о её несовершенности.

Дальнейшее развитие системы может включать усовершенствование существующего алгоритма, в частности написание системы универсальной бинаризации изображения для любой погоды и освещённости. Или пересмотр архитектуры и подхода к анализу данных, к примеру на основе машинного обучения.

Список литературы

1. «Computer Vision: A Modern Approach» David A. Forsyth, Jean Ponce
2. OpenCV official documentation - https://docs.opencv.org/
3. «Learning OpenCV: Computer Vision with the OpenCV Library» Gary Bradski, Adrian Kaehler
4. https://habr.com/ru
5. http://robocraft.ru/blog/computervision/
6. https://medium.com/

Приложение

```
import sys
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import cv2
%matplotlib inline

import os
from itertools import chain
from copy import deepcopy

path = 'results/.'

# A fixed polygon coordinates for the region of interest
ROI_VERTICES = np.array([[(50, 540), (420, 330), (590, 330),
                  (960 - 50, 540)]], dtype=np.int32)

def to_opencv_hsv(*hsv):
   return (hsv[0] / 2, hsv[1] / 100 * 255, hsv[2] / 100 * 255)

# White and yellow color thresholds for lines masking.
# Optional "kernel" key is used for additional morphology
# WHITE_LINES = { 'low_th': to_opencv_hsv(0, 0, 80),
#                 'high_th': to_opencv_hsv(359, 10, 100) }

# YELLOW_LINES = { 'low_th': to_opencv_hsv(49, 10, 100),
#                  'high_th': to_opencv_hsv(57, 18, 100),
#                  'kernel': np.ones((3,3),np.uint64)}

WHITE_LINES = { 'low_th': to_opencv_hsv(0, 0, 80),
                'high_th': to_opencv_hsv(359, 10, 100) }

YELLOW_LINES = { 'low_th': to_opencv_hsv(35, 20, 30),
                 'high_th': to_opencv_hsv(65, 100, 100),
                 'kernel': np.ones((3,3),np.uint64)}
```

```
class Lane(object):

    CRITICAL_SLOPE_CHANGE = 0.1
    MOSTLY_HORIZONTAL_SLOPE = 0.4
    MAX_SLOPE_DIFFERENCE = 0.8
    MAX_DISTANCE_FROM_LINE = 20
    BUFFER_FRAMES = 10
    COLORS = {
        'lane_color': (255,0,255),
        'region_stable': (0,80,60),
        'region_unstable': (255,80,60),
        'left_line': (60,40,220),
        'right_line': (255,0,255)
    }
    THICKNESS = 5
    FIRST_FRAME_LINE_RANGES = {'left_line': range(480),
                               'right_line': range(480,960)}

    # A decision matrix for updating a lane line in order to keep it steady.
    # A weighted average of average lane position from buffer and from the current
frame.
    # 0.1 * frame position + 0.9 * avg from buffer: in case of unstable lane.
    # 1 * frame position + 0 * buffer: in case of stable lane.
    DECISION_MAT = [[.1,.9],[1,0]]

    left_line = None
    right_line = None

    @staticmethod
    def lines_exist():
        return all([Lane.left_line, Lane.right_line])

    @staticmethod
    def fit_lane_line(segments):
```

```
        x, y = [], []

        for line in segments:
            if line.candidate:
                x_coords = list(range(line.x1, line.x2, 1))
                y_coords = list(map(line.get_y_coord, x_coords))
                x.extend(x_coords)
                y.extend(y_coords)

        if x != [] and not Lane.lines_exist():
            lane_line = segments[0].lane_line
            coords = np.array([[x, y] for x,y in zip(x, y)
                               if x in Lane.FIRST_FRAME_LINE_RANGES[lane_line]])
            x = coords[:,0]
            y = coords[:,1]
        if x != []:
            poly_coeffs = np.polyfit(x, y, 1)
            return poly_coeffs, list(zip(x, y))
        else: return None, None

    @staticmethod
    def update_vanishing_point(left, right):
        equation = left.coeffs - right.coeffs
        x = -equation[1] / equation[0]
        y = np.poly1d(left.coeffs)(x)
        x, y = map(int, [x, y])
        left.vanishing_point = [x, y]
        right.vanishing_point = [x, y]

    @staticmethod
    def purge():
        Lane.left_line = None
        Lane.right_line = None

    def __init__(self, segments):

        buffer_frames = Lane.BUFFER_FRAMES
```

```
        # Lane coefficients from the current image
        self.current_lane_line_coeffs, self.points = Lane.fit_lane_line(segments)

        if self.current_lane_line_coeffs is None:
            raise Exception('Cannot initialize lane. No lines detected.')

        # Buffer for lane line smoothing
        self.buffer = np.array(buffer_frames * [self.current_lane_line_coeffs])

        # Publicly available coefficients of lane line
        self.coeffs = self.buffer[0]

        # Stability flag. Set to False if the slope changes too rapidly
        self.stable = True

        # Hough lines which belong to this lane line
        self.segments = None

        # List of points which belong to this lane line. Transformed from segments
        self.points = None

        # Coordinates for drawing this lane line
        self.x1, self.x2, self.y1, self.y2 = 0,0,0,0

    @property
    def a(self):
        """
        Slope of the lane line. Not intended for higher order polynomials.
        """
        if len(self.coeffs) > 2:
            return Exception("You have a higher order polynomial for Lane, but you
treat it as a line.")
        return self.coeffs[0]

    @property
    def b(self):
```

```
        """
        Intercept of the lane line. Not intended for higher order polynomials.
        """
        if len(self.coeffs) > 2:
            return Exception("You have a higher order polynomial for Lane, but you
treat it as a line.")
        return self.coeffs[1]

    # The main client method for dealing with lane updates
    def update_lane_line(self, segments):
        average_buffer = np.average(self.buffer, axis=0)
        self.coeffs = np.average(self.buffer, axis=0)
        self.update_current_lane_line_coeffs(segments)
        weights = Lane.DECISION_MAT[self.stable]
        current_buffer_coeffs = np.dot(weights,
np.vstack([self.current_lane_line_coeffs, average_buffer]))
        self.buffer = np.insert(self.buffer, 0, current_buffer_coeffs, axis=0)[:-1]
        self.update_lane_line_coords()

    def update_current_lane_line_coeffs(self, segments):
        lane_line_coeffs, points = Lane.fit_lane_line(segments)
        if lane_line_coeffs is None:
            lane_line_coeffs = np.average(self.buffer, axis=0)
        if points is not None:
            self.points = points
        average_buffer = np.average(self.buffer, axis=0)
        buffer_slope = average_buffer[0]
        current_slope = lane_line_coeffs[0]
        self.current_lane_line_coeffs = lane_line_coeffs
        if abs(current_slope - buffer_slope) > Lane.CRITICAL_SLOPE_CHANGE:
            self.stable = False
        else: self.stable = True

    def update_segments_list(self, segments):
        self.segments = segments

    def get_x_coord(self, y):
```

```
        return int((y - self.coeffs[1]) / self.coeffs[0])

    def update_lane_line_coords(self):
        # Offset to distinguish lines
        visual_offset = 20
        self.y1 = image.shape[1]
        self.x1 = self.get_x_coord(self.y1)
        self.y2 = self.vanishing_point[1] + visual_offset
        self.x2 = self.get_x_coord(self.y2)

class Line(object):

    def __init__(self, x1, y1, x2, y2):
        if x1 > x2: (x1, y1), (x2, y2) = (x2, y2), (x1, y1)
        self.x1, self.y1 = x1, y1
        self.x2, self.y2 = x2, y2
        self.a = self.compute_slope()
        self.b = self.compute_intercept()
        self.lane_line = self.assign_to_lane_line()



    def __repr__(self):
        return 'Line: x1={}, y1={}, x2={}, y2={}, a={}, b={}, candidate={},
line={}'.format(
                self.x1, self.y1, self.x2, self.y2, round(self.a,2),
                round(self.b,2), self.candidate, self.lane_line)

    def get_coords(self):
        return (self.x1, self.y1, self.x2, self.y2)

    def get_x_coord(self, y):
        return int((y - self.b) / self.a)

    def get_y_coord(self, x):
        return int(self.a * x + self.b)

    def compute_slope(self):
```

```
        return (self.y2 - self.y1) / (self.x2 - self.x1)

    def compute_intercept(self):
        return self.y1 - self.a * self.x1

    @property
    def candidate(self):
        """
        1. The line cannot be horizontal and should have a reasonable slope.
        2. The difference between lane line's slope and this hough line's cannot be too
high.
        3. The hough line should not be far from the lane line it belongs to.
        4. The hough line should be below the vanishing point.
        """
        if abs(self.a) < Lane.MOSTLY_HORIZONTAL_SLOPE: return False
        lane_line = getattr(Lane, self.lane_line)
        if lane_line:
            if abs(self.a - lane_line.coeffs[0]) > Lane.MAX_SLOPE_DIFFERENCE:
return False
            if self.distance_to_lane_line > Lane.MAX_DISTANCE_FROM_LINE:
return False
            if self.y2 < Lane.left_line.vanishing_point[1]: return False
        return True

    def assign_to_lane_line(self):
        if self.a < 0.0: return 'left_line'
        else: return 'right_line'

    @property
    def distance_to_lane_line(self):

        lane_line = getattr(Lane, self.lane_line)
        if lane_line is None: return None
        avg_x = (self.x2 + self.x1) / 2
        avg_y = (self.y2 + self.y1) / 2
        distance = abs(lane_line.a * avg_x - avg_y +
                lane_line.b) / math.sqrt(lane_line.a ** 2 + 1)
```

```
        return distance

def get_lane_lines_mask(hsv_image, colors):

    masks = []
    for color in colors:
        if 'low_th' in color and 'high_th' in color:
            mask = cv2.inRange(hsv_image, color['low_th'], color['high_th'])
            if 'kernel' in color:
                mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, color['kernel'])
            masks.append(mask)
        else: raise Exception('High or low threshold values missing')
    if masks:
        return cv2.add(*masks)

def hough_line_transform(image, rho, theta, threshold, min_line_length,
max_line_gap):

    lines = cv2.HoughLinesP(image, rho, theta, threshold, np.array([]),
                        minLineLength=min_line_length, maxLineGap=max_line_gap)
    if lines is not None:
        filtered_lines = list(filter(lambda l: l.candidate, map(lambda line:
Line(*line[0]), lines)))
        return filtered_lines
    else: return None

def update_lane(segments):
    if segments is not None:
        left = [segment for segment in segments if segment.lane_line == 'left_line']
        right = [segment for segment in segments if segment.lane_line == 'right_line']
        if not Lane.lines_exist():
            Lane.left_line = Lane(left)
            Lane.right_line = Lane(right)
        Lane.update_vanishing_point(Lane.left_line, Lane.right_line)
        Lane.left_line.update_lane_line([l for l in left if l.candidate])
        Lane.right_line.update_lane_line([r for r in right if r.candidate])
```

```
#     width = 960
#     height = 540
#     frame = cv2.resize(image, (960,540))
#     return frame

def image_pipeline(image):
    """
    Main image pipeline with 3 phases:
    * Raw image preprocessing and noise filtering;
    * Lane lines state update with the information gathered in preprocessing phase;
    * Drawing updated lane lines and other objects on image.
    """
    image = cv2.resize(image, (960,540))

    ### Phase 1: Image Preprocessing
    hsv_image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)

    binary_mask = get_lane_lines_mask(hsv_image, [WHITE_LINES,
YELLOW_LINES])

    binary_mask = cv2.rectangle(binary_mask,(0,0),(960,270),(0,0,255),-1)

    masked_image = draw_binary_mask(binary_mask, hsv_image)

#     cv2.imwrite(os.path.join(path , 'masked0.jpg'), masked_image)
    blank_image = np.zeros_like(image)

#     cv2.imwrite(os.path.join(path , 'masked2.jpg'), blank_image)
    edges_mask = canny(masked_image, 280, 360)

#     cv2.imwrite(os.path.join(path , 'masked.jpg'), edges_mask)
    if not Lane.lines_exist():
        edges_mask = region_of_interest(edges_mask, ROI_VERTICES)

    edges_image = draw_canny_edges(edges_mask, blank_image)

    segments = hough_line_transform(edges_mask, 1, math.pi / 180, 5, 5, 8)
```

```
    ### Stage 2: Lane lines state update

    update_lane(segments)

    ### Stage 3: Drawing

    # Snapshot 1
#     out_snap1 = np.zeros_like(image)
#     out_snap1 = draw_binary_mask(binary_mask, out_snap1)
#     out_snap1 = draw_filtered_lines(segments, out_snap1)
#     snapshot1 = cv2.resize(deepcopy(out_snap1), (240,135))
#     path = '.'
#     cv2.imwrite(os.path.join(path , 'binary_mask_with_line.jpg'), out_snap1)
    out_snap0 = np.zeros_like(image)
    out_snap0 = draw_binary_mask(binary_mask, out_snap0)
    out_snap1 = deepcopy(image)
    out_snap1 = draw_filtered_lines(segments, out_snap1)
    snapshot1 = cv2.resize(deepcopy(out_snap1), (240,135))
    cv2.imwrite(os.path.join(path , 'binary_mask_with_line.jpg'), out_snap1)
    # Snapshot 2
    out_snap2 = np.zeros_like(image)
    out_snap2 = draw_canny_edges(edges_mask, out_snap2)
    out_snap3 = out_snap2
    out_snap2 = draw_points(Lane.left_line.points, out_snap2,
Lane.COLORS['left_line'])
    out_snap2 = draw_points(Lane.right_line.points, out_snap2,
Lane.COLORS['right_line'])
    out_snap2 = draw_lane_polygon(out_snap2)
    snapshot2 = cv2.resize(deepcopy(out_snap2), (240,135))

    cv2.imwrite(os.path.join(path , 'snapshot2.jpg'), out_snap2)



    cv2.imwrite(os.path.join(path , 'snapshot3.jpg'), out_snap3)
    # Augmented image
    output = deepcopy(image)
```

```
    output = draw_filtered_lines(segments, output)
    plt.imshow(snapshot1)
    plt.imshow(snapshot2)



    output = deepcopy(image)
    output = draw_lane_lines([Lane.left_line, Lane.right_line], output,
shade_background=True)
    return output

# Simple drawing routines to draw figures on image.

# Though cv2 functions mutate objects in arguments,
# all routines below explicitly return a 3-channel RGB image.

# Basic signature: draw_some_object(what_to_draw,
background_image_to_draw_on, kwargs)

def draw_binary_mask(binary_mask, img):
    if len(binary_mask.shape) != 2:
        raise Exception('binary_mask: not a 1-channel mask. Shape:
{}'.format(str(binary_mask.shape)))
    masked_image = np.zeros_like(img)
    for i in range(3):
        masked_image[:,:,i] = binary_mask.copy()
    return masked_image

def draw_canny_edges(binary_mask, img):
    return draw_binary_mask(binary_mask, img)

def draw_filtered_lines(lines, img, color=[255, 0, 0], thickness=2):
    """
    Uses the output of `hough_line_transform` function to draw lines on an image.
    """
    if lines is None: return img
    line_img = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)
    for line in lines:
```

```
            x1,y1,x2,y2 = line.get_coords()
            cv2.line(line_img, (x1, y1), (x2, y2), color, thickness)
        return weighted_img(line_img, img)

    def draw_points(points, img, color=(255,255,0)):
        if points is None: return img
        for point in points:
            cv2.circle(img, point, 2, color, -1)
        return img

    def draw_lane_lines(lane_lines, img, shade_background=False):
        if shade_background: α = 0.8
        else: α = 1.
        lane_line_image = np.zeros_like(img)
        for line in lane_lines:
            line.update_lane_line_coords()
            cv2.line(lane_line_image, (line.x1, line.y1), (line.x2, line.y2),
                    Lane.COLORS['lane_color'], Lane.THICKNESS)
        return weighted_img(lane_line_image, img, α=α, β=1.)

    def draw_lane_polygon(img):
        offset_from_lane_edge = 20
        color = Lane.COLORS['region_stable']

        if not Lane.lines_exist(): return img

        # Polygon points
        p1 = [Lane.left_line.x1,Lane.left_line.y1]
        p2 = [Lane.left_line.get_x_coord(Lane.left_line.y2 + offset_from_lane_edge),
                                Lane.left_line.y2 + offset_from_lane_edge]
        p3 = [Lane.right_line.get_x_coord(Lane.left_line.y2 + offset_from_lane_edge),
                                 Lane.right_line.y2 + offset_from_lane_edge]
        p4 = [Lane.right_line.x1,Lane.right_line.y1]

        polygon_points = np.array([p1, p2, p3, p4], np.int32).reshape((-1,1,2))

        if not Lane.left_line.stable or not Lane.right_line.stable:
```

```
            color = Lane.COLORS['region_unstable']

        poly_img = np.zeros_like(img)
        cv2.fillPoly(poly_img,[polygon_points], color)
        return weighted_img(img, poly_img)

    def draw_dashboard(img, snapshot1, snapshot2):
        # TODO: refactor this
        if not Lane.lines_exist(): return img
        cv2.CV_FILLED = -1
        image_copy = deepcopy(img)
        cv2.rectangle(image_copy, (0,0), (540,175), (0,0,0), cv2.CV_FILLED)
        img = weighted_img(image_copy, img, α=0.3, β=0.7)
        img[20:155,20:260,:] = snapshot1
        img[20:155,280:520,:] = snapshot2
        return img

    def draw_on_gray_with_color_mask(img, binary_mask):
        """
        Returns a gray-ish image with colorized parts described in binary_mask.
        img should be a 3-channel image.
        """
        image_gray = grayscale(img)
        mask = np.zeros_like(img)
        color_mask = cv2.bitwise_and(img, img, mask= binary_mask)
        binary_mask_inv = cv2.bitwise_not(binary_mask)
        image_gray = cv2.bitwise_and(image_gray, image_gray,
    mask=binary_mask_inv)

        output = np.zeros_like(img)
        for i in range(3):
            output[:,:,i] = image_gray
        output = cv2.add(output, color_mask)
        return output

    import math
```

```
def grayscale(img):

    return cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    # Or use BGR2GRAY if you read an image with cv2.imread()
    # return cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

def canny(img, low_threshold, high_threshold):
    """Applies the Canny transform"""
    return cv2.Canny(img, low_threshold, high_threshold)

def gaussian_blur(img, kernel_size):
    """Applies a Gaussian Noise kernel"""
    return cv2.GaussianBlur(img, (kernel_size, kernel_size), 0)

def region_of_interest(img, vertices):
    """
    Applies an image mask.

    Only keeps the region of the image defined by the polygon
    formed from `vertices`. The rest of the image is set to black.
    """
    #defining a blank mask to start with
    mask = np.zeros_like(img)

    #defining a 3 channel or 1 channel color to fill the mask with depending on the
input image
    if len(img.shape) > 2:
        channel_count = img.shape[2]  # i.e. 3 or 4 depending on your image
        ignore_mask_color = (255,) * channel_count
    else:
        ignore_mask_color = 255

    #filling pixels inside the polygon defined by "vertices" with the fill color
    cv2.fillPoly(mask, vertices, ignore_mask_color)

    #returning the image only where mask pixels are nonzero
    masked_image = cv2.bitwise_and(img, mask)
```

```
    return masked_image


def draw_lines(img, lines, color=[255, 0, 0], thickness=2):

    for line in lines:
        for x1,y1,x2,y2 in line:
            cv2.line(img, (x1, y1), (x2, y2), color, thickness)

def hough_lines(img, rho, theta, threshold, min_line_len, max_line_gap):

    lines = cv2.HoughLinesP(img, rho, theta, threshold, np.array([]),
minLineLength=min_line_len, maxLineGap=max_line_gap)
    line_img = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)
    draw_lines(line_img, lines)
    return line_img

def weighted_img(img, initial_img, α=0.8, β=1., λ=0.):

    return cv2.addWeighted(initial_img, α, img, β, λ)

image = mpimg.imread('test_images/road222.jpg')
Lane.purge()
plt.figure(figsize=(12,8))
cv2.imwrite(os.path.join(path , 'result.jpg'), image_pipeline(image))
plt.imshow(image_pipeline(image))
```