Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»
Направление «Информатика и вычислительная техника»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой,
_____ Г.И. Радченко
«___»_____ 2018 г.

Разработка прототипа системы автопилотирования для автомобилей

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ – 09.03.01.2018.151.ПЗ ВКР

Руководитель работы,
доцент каф. «Электронные
вычислительные машины»
_____ Ю.Г. Плаксина
«___»_____ 2018 г.

Автор работы
студент группы КЭ-484
_____ Г.И. Чистохина

«___»_____ 2018 г.

Нормоконтролер,
старший преподаватель
_____ В. В. Лурье
«___»_____ 2018 г.

Челябинск 2018

АННОТАЦИЯ

Чистохина Г.И. Разработка прототипа системы автопилотирования для автомобилей – Челябинск: ЮУрГУ, ВШЭКН; 2018, 64 с. 8 ил., библиогр. список – 20 наим.

В выпускной квалификационной работе представлена разработка прототипа системы автопилотирования для автомобилей, в базовой сборке которых не включена данная система.

Проведен обзор программных средств для разработки программного обеспечения, и на основе требований к разрабатываемой системе были выбраны средства разработки и язык программирования.

Созданы тестовая модель и прототип программного продукта, проведено установление соответствия предъявленным требованиям.

					ЮУрГУ-09.03.01.2018.151 ПЗ ВКР			
Изм	*Лист*	*№ докум.*	*Подп..*	*Дата*				
Разраб.		*Г.И. Чистохина*			*Разработка прототипа системы автопилотирования для автомобилей*	*Лит.*	*Лист*	*Листов*
						Д	*3*	*64*
Пров.		*Ю.Г. Плаксина*				*ФГБОУ ВПО «ЮУрГУ» (НИУ) Кафедра ЭВМ*		
Н.контр.		*В.В. Лурье*						
Утв.		*Г.И. Радченко*						

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время происходит формирование полноценной экосистемы потребителей и поставщиков услуг, систем и современных транспортных средств на основе интеллектуальных платформ, сетей и инфраструктуры в логистике людей и вещей.

Все больше компаний разрабатывают технологии для транспортных средств, посредством которых они могут общаться друг с другом и использовать данные в реальном времени от объектов производственной инфраструктуры. Особое положение в области изучения транспортных систем занимает разработка научных основ проектирования мобильных транспортных роботов, автопилотируемых автомобилей и роботизированных транспортных комплексов различного назначения, отвечающих технологическим, конструктивным требованиям и структуре управления производством.

На сегодняшний день разработаны автопилотируемые автомобили, системы автоматического управления машиной, способные справится с поездкой по дороге без помощи человека. Они могут получать информацию о своем местоположение и окружающей среде от внешних датчиков (тактильных, акустических, визуальных, локационных, температурных, химических). Но все эти разработки применяются при базовой сборке автомобиля или системы, их невозможно внедрить в уже собранный автомобиль.

1. ПОСТАНОВКА ЗАДАЧИ

1.1 Цель выпускной квалификационной работы

Целью выпускной квалификационной работы является разработка прототипа системы автопилотирования для автомобилей, ориентированного на использование в автомобилях, в базовой комплектации которых не установлен автопилот.

1.2 Задачи выпускной квалификационной работы

Для достижения выделенной цели разработки были поставлены следующие задачи, требующие решения:

- ознакомиться со сферой машинного обучения и компьютерного зрения;
- провести сравнительный анализ аналогов, выделить преимущества и недостатки;
- составить техническое задание на разрабатываемый продукт;
- на основе требований разработать архитектуру программного комплекса;
- используя разработанную архитектуру, создать тестовую модель;
- в течении разработки и по её завершению оценить соответствие программного продукта техническим требованиям;

1.3 Конечный результат выпускной квалификационной работы

Конечным результатам выполнения выпускной квалификационной работы будет являться тестовая модель – прототип системы автопилотирования для автомобилей.

2. ОБЗОР ЛИТЕРАТУРЫ

2.1 Обзор литературы по теме выпускной квалификационной работы

Система классификации автопилотирования, основанная на шести разных уровнях (от полностью ручной до полностью автоматизированных систем), была опубликована в 2014 году SAE International, органом по стандартизации автомобилей, как J3016, таксономия и определения для терминов, относящихся к автоматизированным системам управления автомобилем на дороге [18] [19]. Эта система классификации основана на количестве вмешательства водителя и внимательности, а не на возможностях транспортного средства. В Соединенных Штатах в 2013 году Национальная администрация безопасности дорожного движения (NHTSA) выпустила официальную систему классификации [20], но отказалась от этой системы в пользу стандарта SAE в 2016 году. Также в 2016 году SAE обновила свою классификацию под названием J3016_201609 [20].

В данной классификации различается 5 уровней автопилотирования:

Уровень 0: на этом уровне происходят автоматические оповещения о проблемах с системой и имеется возможность мгновенного вмешательства (парктроник), но на этом уровне автопилот не имеет устойчивого управления транспортным средством.

Уровень 1: его называют «руки». Здесь взаимодействуют водитель и автоматический контроль доступа к автомобильной системе. Примерами являются Adaptive Cruise Control (ACC), где водитель управляет рулевым управлением, а автоматическая система контролирует скорость; также помощь при парковке при автоматическом рулевом управлении, а скорость - при ручном управлении. Водитель должен быть готов вернуться в полный контроль в любое время.

Уровень 2: «руки выключены». Автоматическая система полностью управляет автомобилем (ускорение, торможение и рулевое управление). Водитель должен следить за вождением и быть готовым немедленно вмешаться в любое время, если автоматическая система не может правильно ответить. Сокращение «руки» не следует воспринимать буквально.

Фактически, контакт между рукой и колесом часто является обязательным во время вождения SAE 2, чтобы подтвердить, что водитель готов вмешаться.

Уровень 3: «глаз выключен». Водитель может смело отвлечь свое внимание от задач управления, например, может читать или смотреть фильм. Автомобиль будет обрабатывать ситуации, требующие немедленного реагирования, например, экстренного торможения. Водитель должен быть готов вмешаться в течение ограниченного времени, указанного изготовителем. Например, Audi A8 Luxury Sedan 2018 был первым коммерческим автомобилем, который, как утверждается, мог работать на автопилоте третьего уровня. У этого автомобиля есть так называемый пилот-зонд. При активации человеком-водителем автомобиль полностью контролирует все аспекты движения в замедленном темпе со скоростью до 60 километров в час (37 миль / ч). Функция работает только на автомагистралях с физическим барьером, разделяющим один поток трафика от встречного трафика.

Уровень 4: «mind off». Он как уровень 3, но для безопасности внимание водителя никогда не требуется, то есть водитель может спокойно спать или покидать сиденье водителя. Самообслуживание поддерживается только в ограниченных пространственных зонах (geof.) Или при особых обстоятельствах, таких как пробки. Вне этих областей или обстоятельств транспортное средство должно быть в состоянии безопасно остановить поездку, то есть припарковать автомобиль, если водитель не получит контроль.

Уровень 5: «дополнительное рулевое колесо». Вмешательство человека вовсе не требуется. Примером может служить роботизированное такси.

Разработанный прототип системы автопилотирования можно отнести ко второму уровню. Автоматическая система полностью управляет тестовой моделью, но, если автоматическая система не может правильно ответить, водитель должен быть готовым вмешаться.

В системе управления автопилотируемой модели алгоритм прокладки направления движения (управление маршрутом) занимает одно из основных

мест, цель которого заключается в формировании необходимого плана действии по перемещениям модели, с учетом имеющейся в наличии информации об окружающей среде. Необходимо выдерживать направление движения, в том числе движение в местах пересечения или разветвления маршрута, обеспечивать безопасность движения (предотвращать наезды и столкновения с препятствиями) и требуемую точность положения модели относительно объекта в момент остановки. Определять и контролировать местоположение тестовой модели на маршруте. Информация о местоположении поступает посредством особых меток, расположенных по маршруту, которые считываются программой системы управления. Метки могут быть активного и пассивного типа. Используется два управляемых параметра режима работы модели на маршруте – ускорение и скорость. Обе функции программируются в зависимости от загрузки тестовой модели и сложности конфигурации маршрута.

При помощи алгоритмов глубокого машинного обучения обучается нейросеть, которая по мгновенно получаемому визуальному образу с камеры классифицирует полученные изображения и уже на основе изображения, прототип позиционирует себя в производственном помещении.

Обобщённая функциональная схема прототипа автопилотируемой системы обычно включает следующие элементы [1] (рис.1):

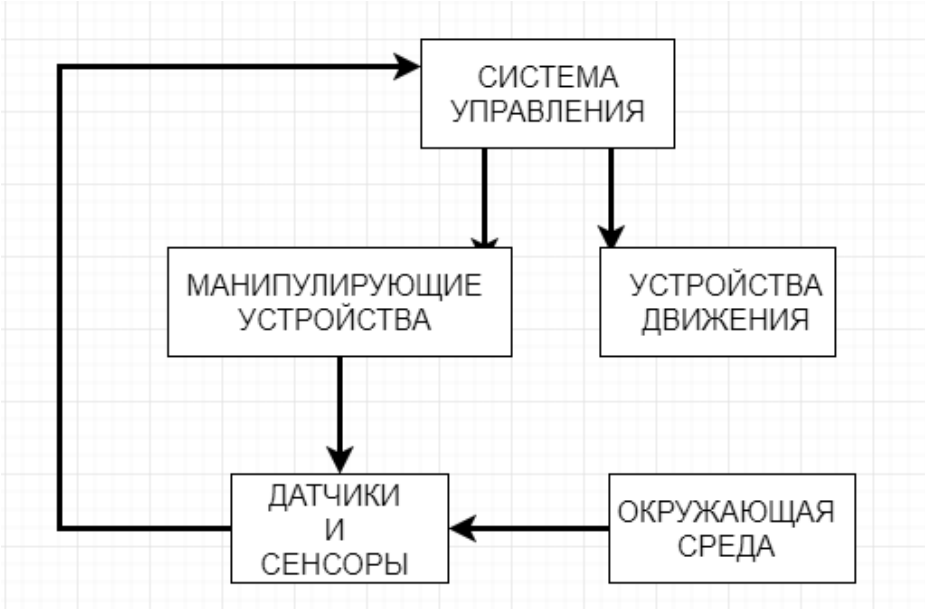

Рисунок 1. Обобщеная функциональная схема транспортного робота

Для обучения прототипа необходимо было решить задачу классификации. Входными данными являлись изображения. Поэтому рассматривались такие методы обучения нейронной сети, как метод k ближайших соседей, однослойный персептрон и персептрон.

Однослойный персептрон – линейный алгоритм, основнный на модели нервной клетки. Представляет собой пример нейронной сети с одним скрытым слоем. Значения признаков рассматриваются как импульсы, поступающие на вход нейтрона. Если суммарный импульс при=евышает порог активации, то нейрон возбуждается и выдает на выходе 1, иначе выдается 0. Однако в данном проекте этот алгоритм допускает около 50% ошибок, т.к входные данные линейно неразделимы.

Персептон – это обучение с коррекцией ошибки. После обучения перцептрон готов работать в режиме распознавания или обобщения В этом режиме перцептрону предъявляются «не знакомые» перцептрону объекты, и перцептрон должен установить, к какому классу они принадлежат. Работа перцептрона состоит в следующем: при предъявлении объекта возбудившиеся A — элементы передают сигнал R — элементу, равный сумме соответствующих коэффициентов w_i. Если эта сумма положительна, то принимается решение, что данный объект принадлежит к первому классу, а если она отрицательна — то второму. Представляет собой метод обучения, при котором вес связи не изменяется до тех пор, пока текущая реакция персептона остается правильной. При появлении неаправильной реакции вес изменяется на единицу, а знак (+/-) определяется противоположным знаком ошибки.

Метод k блиайших соседей - в процессе обучения алгоритм запоминает все векторы признаков и соответствующие им метки классов. При работе с реальными данными, т.е. наблюдениями, метки класса которых неизвестны, вычисляется расстояние между вектором нового наблюдения и ранее запомненными. Затем выбирается kk ближайших к нему векторов, и новый

объект относится к классу, которому принадлежит большинство из них. Несмотря на свою относительную алгоритмическую простоту, метод показывает хорошие результаты. Главным его недостатком является высокая вычислительная трудоемкость, которая увеличивается квадратично с ростом числа записей в наборе данных.

2.2 Обзор литературы по обоснованию выбора программного обеспечения

В данном проекте должен производиться анализ изображения, предобработка данных, а затем векторизация в понятный для алгоритмов распознавания вид. Для данной задачи используется OpenCV – фреймворк для работы с изображениями на пиксельном уровне с помощью матричных операций [13]. Библиотека OpenCV направлена как раз на обеспечение работы инструментов для устранения описанных выше проблем компьютерного зрения. OpenCV (Open Source Computer Vision Library) – это библиотека компьютерного зрения, которая поставляется с открытым исходным программным кодом. Спектр возможностей данной библиотеки очень широк. В ней собрано большое количество алгоритмов для использования технологий компьютерного зрения. После подключения данной библиотеки к своему проекту пользователь получает доступ более чем к 500 функций, предназначенных для решения разнообразных задач. Помимо алгоритмов для работы с технологиями компьютерного зрения, данная библиотека применяется и для обработки изображений, содержит большое число численных алгоритмов и многое другое. Главной целью библиотеки OpenCV является предоставление легкого в использовании интерфейса, который поможет облегчить использование технологий компьютерного зрения в довольно сложных приложениях. Функции, которые поддерживает библиотека, охватывают разнообразные сферы компьютерного зрения, от медицины, безопасности и до стереозрения и робототехники. Все это благодаря тому, что компьютерное зрение и машинное обучение — два неразрывно связанных понятия. Кроме того, библиотека OpenCV содержит библиотеку MLL, с английского Machine Learning Library. Даная библиотека является библиотекой общего назначения и ориентирована на распознавание статических образов и технологию кластеризации. Данная библиотека является очень эффективной для решения задач компьютерного зрения, которое как раз и является основой OpenCV. Однако для решения конкретных задач машинного обучения данная библиотека не приспособлена и является

довольно обобщенной. Помимо библиотеки OpenCV, существуют также и другие библиотеки компьютерного зрения. Например, библиотеки Matrox Imaging Library, Camellia 15 Library, Open eVision, HALCON, VXL, libCVD, IVT, LTI, AForge.NET и некоторые другие.

AForge.NET является библиотекой с открытым исходным кодом, созданной на языке C#, которая предназначена для разработчиков и исследователей в области компьютерного зрения. Кроме того, в библиотеке есть функционал для разработчиков в области искусственного интеллекта. Спектр возможностей библиотеки довольно широк: обработка изображений, нейронные сети, генетические алгоритмы, нечеткая логика, машинное обучение, робототехника и многое другое. Библиотека включает несколько основных компонентов. AForge.Imaging – библиотека подпрограмм для обработки изображений и фильтров. AForge.Vision – библиотека компьютерного зрения. AForge.Video – набор библиотек для работы с видео информацией. AForge.Neuro – библиотека для выполнения разнообразных действий и операций с нейронными сетями. AForge.Genetic – библиотека подпрограмм для использования генетических алгоритмов для решения различных задач. AForge.Fuzzy – библиотека для работы с нечеткой логикой. AForge.Robotics – библиотека, обеспечивающая поддержку некоторых методов, применяемых в сфере робототехники. AForge.MachineLearning – библиотека для работы с элементами машинного обучения. AForge.NET постоянно улучшается и прогрессирует. По данной библиотеке есть большое количество примеров, демонстрирующих ее работу, а также актуальная html-документация, которая помогает начинающим разработчикам, которые хотят применять данный фреймворк в своих проектах. Кроме того, как и у библиотеки OpenCV существует сообщество, где можно задавать вопросы и делиться своими наработками по функциям и компонентам AForge.NET. Однако количество участников сообщества по данной библиотеке все-таки меньше, чем в сообществах по библиотеке OpenCV. Данный фреймворк уступает по популярности OpenCV. Еще одним недостатком является тот факт, что вся документация по библиотеке написана только на английском языке,

поэтому у некоторых разработчиков могут возникнуть трудности с изучением и работой с данной библиотекой.

VXL, от английского the Vision-something-Libraries, – это набор библиотек, написанных на языке C++, которые предназначены для научных исследований и реализации технологий компьютерного зрения. VXL была написана в ANSI/ISO C++ и предназначена для портативных платформ. Библиотека состоит из нескольких основных составляющих: VNL (числа) – численные алгоритмы и контейнеры, например, матрицы, векторы, оптимизаторы и т.д., VIL (изображения) – загрузка, сохранение и редактирование изображений во многих наиболее распространенных форматах (также существует возможность работы с очень большими изображениями), VGL (геометрия) – геометрия точек, кривых и других элементарных объектов в одно-, двух- и трехмерном пространствах, VSL (входный и выходной потоки), VBL (основные шаблоны), VUL (утилиты) – разный функционал для независимых платформ. Кроме основных библиотек, входящих в состав VXL, есть также и дополнительные, которые отвечают за такие понятия, как численные алгоритмы, обработка изображений, системы координат, геометрия камеры, стерео, манипуляции с видео потоком, восстановление структуры при движении камеры, графический дизайн, функции отслеживания, топология, классификаторы, 3d визуализация и многое другое. Особенность библиотеки заключается в том, что каждый ее компонент может использоваться отдельно, не ссылаясь на другие компоненты библиотеки. Таким образом, в приложении можно использовать только те библиотеки, которые действительно необходимы. VXL используется по всему миру. Библиотека применяется в сфере обучения и промышленности, некоторые ведущие мировые эксперты в сфере компьютерного зрения пользуются данной библиотекой. Существует документация по VXL с описанием каждого класса и функций. Однако недостатком является тот факт, что аналогично AForge.NET вся документация написана на английском языке.

LTI или LTI-lib является объектно-ориентированной библиотекой алгоритмов и структур данных, часто применяется при обработке изображений

и в сфере компьютерного зрения. LTI-lib была разработана в техническом университете как часть научно-исследовательских проектов в области компьютерного зрения с технологиями робототехники, распознавания объектов, голоса и жестов. Основной целью разработки данной библиотеки является создание объектно- ориентированной библиотеки на языке C++, что во многом упрощало бы использование кода и его обслуживание, но при этом были бы обеспечены быстрые алгоритмы, которые можно было бы использовать в реальных приложениях. Библиотека была разработана с применением GCC (набор компиляторов, применяемый для разнообразных языков программирования) под Linux и Visual C++ по Windows NT. Многие классы инкапсулируют Windows/Linux функциональность для того, чтобы упростить решение системных или аппаратныхзадач (например, классы для многопоточности и синхронизации, измерения времени и доступ к последовательному порту). Остальные классы, количество которых превышает 500, выполняют одну из следующих функций. Функции линейной алгебры: предоставление матриц, векторов, решения линейных уравнений, собственные векторы, расчет статистики и многое другое. Функции классификации кластеризации: радиально базисные функции-классификаторы, методы опорных векторов, метод K-средних, нечеткие C-средства, классификация статистических данных. Функции для обработки изображений: разнообразные подходы к сегментации, линейные фильтры, вейвлет-преобразования и многое другое. Предоставление инструментов для визуализации и рисования. Самое сложное при разработке алгоритмов обработки изображений в C++ это представление временных образов во время отладки. Благодаря объектно-ориентированной архитектуре библиотеки LTI-lib, для предпросмотра необходимо создать объект представления и дать ему образ, который необходимо показать. Документация по библиотеке написана на английском языке. Что касается лицензии, то библиотека LTI, также, как и OpenCV, является свободно распространяемым программным обеспечением. Вся пункты и ограничения по лицензии значатся в GNU Lesser General Public License, кроме того, там же можно посмотреть более подробную информацию

по лицензии. Существует большое количество проектов, разработанных с применением технологий компьютерного зрения, для которых использовалась именно библиотека LTI.

3. АНАЛИЗ АНАЛОГОВ

Внедряемые системы автопилотирования малоизвестны, и информации по данным разработкам очень мало. Поэтому были изучены автопилотируемые автомобили известных компаний, таких как Audi, Tesla Motors, BMW.

В рамках основной презентации NVIDIA на выставке CES компания Audi продемонстрировала интеллектуальные возможности Audi Q7 deep learning concept на трансформируемой открытой площадке, специально созданной для автопилотирования. Автомобиль ориентируется при помощи фронтальной камеры с разрешением 2 Мп, обменивающейся данными с вычислительным модулем NVIDIA Drive PX 2, который, в свою очередь, с высокой точностью осуществляет рулевое управление. Высокоэффективный контроллер создан специально для использования в системах автоматического управления автомобилем. Методы обучения, используемые для Audi Q7 deep learning concept, схожи с методами глубокого обучения с подкреплением (deep reinforcement learning). Здесь нейронные сети обучались определенной практической задаче. Если на конференции NIPS модель, выполненная в масштабе 1:8, училась парковаться путем проб и ошибок, то нейронная сеть Audi Q7 deep learning concept получает конкретные, существенные для концепта данные в ходе тренировочных поездок. То есть она учится у водителя.

У компании BMW в основе программного обеспечения — глубокие нейронные сети, которые специалисты целенаправленно обучали автономному вождению и распознаванию динамических сигналов, регулирующих дорожное движение. Вначале модель под управлением человека знакомилась с маршрутом и окружающей обстановкой, наблюдая и используя для этого дополнительные обучающие камеры. При этом устанавливались взаимосвязи между реакциями водителя и событиями, распознаваемыми при помощи камер. Впоследствии автомобиль стал способен уже самостоятельно понимать указания, подаваемые, например, в виде временного сигнала, регулирующего движение, правильно интерпретировать их и действовать в соответствии с

ситуацией. При появлении соответствующего сигнала концепт-кар немедленно меняет стратегию и выбирает короткий или длинный маршрут. Система спроектирована настолько надежно, что способна справляться даже с такими внешними факторами, как изменяющиеся погодные условия и степень освещения. Она выполняет свои задачи днем и ночью, при прямом солнечном свете и искусственном освещении.

4. РАЗРАБОТКА ТРЕБОВАНИЙ К ПРОГРАММНОМУ КОМЛЕКСУ

4.1. Требования к функциональным характеристикам

4.1.1. Составные части программного комплекса

Программный комплекс должен состоять из следующих частей:

- алгоритмы обработки изображений;
- обученная нейросеть;
- алгоритмы управления мобильного робота.

Алгоритм обработки изображений представляет собой либо подключаемую библиотеку, которая выделяет необходимые цвета, а потом производит бинаризацию, либо готовый проект на выбранном языке программирования.

Обученная нейросеть отвечает за классификацию и распознавание образов, в соответствии с которыми вызываются алгоритмы управления мобильного робота.

4.1.2. Функциональные требования прототипа

Учитывая, что разрабатываемый прототип относится к автопилотированию 2 уровня, то формируется ряд требований. Для автопилотируемого автомобиля необходимо выдерживать направление движения, в том числе движение в местах пересечения или разветвления маршрута, обеспечивать безопасность и требуемую точность положения модели относительно объекта в момент остановки. Определять и контролировать местоположение на маршруте. Информация о местоположении поступает посредством особых меток, расположенных по маршруту, которые считываются программой системы управления. Используется два управляемых параметра режима работы прототипа на маршруте – ускорение и скорость. Обе функции программируются в зависимости от загрузки модели и сложности конфигурации маршрута.

4.1.3 Требования к созданию алгоритма управления тестовой модели

Алгоритм управления тестовой моделью должен представлять собой функцию или метод класса (в зависимости от языка программирования, на котором создается алгоритм принятия решения), запускаемый в каждый такт. Сам алгоритм может храниться в нескольких файлах, вызывать другие размещенные в этом файле функции/методы, использовать собственные локальные переменные и т.п.

4.2 Требования к надежности

При отказе оборудования программный комплекс должен обеспечить возможность продолжения работы с момента прерывания для продолжения пути следования. Если же аппаратная часть сломалась и произошло что-то экстренное, то требуется вмешательство водителя.

4.3. Требования к информационной и программной совместимости

Входными данными программного комплекса является исходный код алгоритма на C++, поскольку на нем удобно работать с датчиками за счет работы библиотек алгоритмов. В комплексах должна быть заложена возможность в дальнейшем расширить область поддерживаемых языков.

Сам программный комплекс может быть исполнен на Python, т.к. на нем существуют готовые решения для работы с компьютерным зрением и машинным обучением.

5. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРИЛОЖЕНИЯ

Для проектирования и разработки программного обеспечения были выделены следующие этапы:

- подбор элементов для тестовой модели;
- построение тестовой модели;
- обработка изображений;
- создание программного кода;
- тестирование.

5.1 Подбор элементов для тестовой модели

В качестве аппаратной части системы управления была выбрана платформа Orange Pi PC One [6]. При проведении анализа существующих аналогов было установлено, что эта платформа имеет более высокие вычислительные возможности, а также хорошо приспособлена к перегреву. Для определения расстояния до объекта к дискретным портам этой платформы были подключены ультразвуковые датчики расстояния HC-SR04. Такие датчики широко используются, имеют невысокую стоимость, а подробности их применения хорошо документированы. Для случаев, когда ультразвуковые датчики не могут определить точное расстояние до объекта к дискретным портам также были подключены ИК-датчики препятствий YL-63 (FC-51). Дополнительно к цифровому порту Orange Pi подключена камера, которая посылает на компьютер кадры в формате изображения для распознавания и анализа объектов, которые она фиксирует. Кроме этого к платформе подключены шаговые моторы, действующие в соответствии с переданными им командами. Благодаря предложенной архитектуре устройство определяет тот или иной объект, способно объехать его, остановиться перед ним (в случае такой необходимости) и ориентироваться в пространстве.

5.2 Построение тестовой модели

На основании проведенного анализа был разработан прототип модели автомобиля (машинки) (рис. 2). Для проверки соответствия архитектуры модели ее назначению (специфике ее работы) были проведены исследовательские испытания. В ходе исследовательских испытаний осуществлялась проверка распознавания прототипом сигнальных знаков на специально разработанном для этих целей полигоне. Критериями качества распознавания являлись угол обзора сигнального знака, расстояние до знака, освещенность.

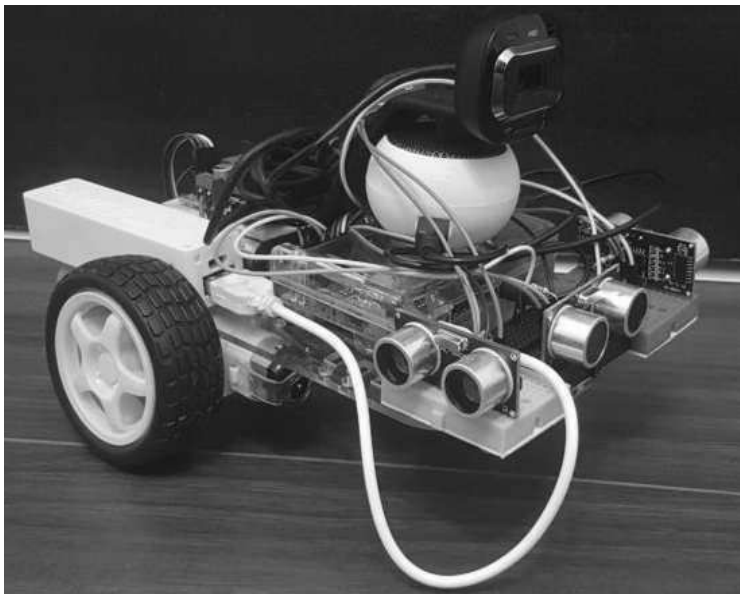

Рисунок 2. Тестовая модель

5.3 Обработка изображений

При получении изображения, на одноплатном компьютере производится его анализ, предобработка данных, а затем векторизация в понятный для алгоритмов распознавания вид.

Исходное RGB-изображение преобразуется в HSV – именно в этой цветовой модели удобно выделять диапазоны конкретных цветов [7]. Для определения графических символов представляют интерес оттенки жёлтого и белого. После перевода изображения в HSV рекомендуется применить размытие по Гауссу [8], но в данном случае оно снизило качество распознавания. Следующая стадия – бинаризация (преобразование

изображения в бинарную маску с цветами, используемыми в сигнальных знаках) [9].

После обработки входных данных система управления вырабатывает сигналы, регулирующие направление движения. Эти сигналы передаются на устройства движения, в качестве которых выступают шаговые двигатели. При распознавании сигнального знака, робот поворачивает. При наличии объекта на пути движения – объезжает его. При невозможности объезда робот ищет другой путь.

В общем виде алгоритм обработки изображения можно представить следующим образом.

Первый этап – получение изображения с камеры. При разработке использовалась web-камера с разрешением 1280x720. OpenCV предоставляет довольно простой способ работы с web-камерами. (модуль HighGui).

Второй этап – преобразование полученного изображения. На втором этапе осуществляется распознавание только некоторых групп сигнальных знаков. В результате формируется фильтр, после применения которого получается темное изображение, где видны только контуры знака.

Третий этап – применение порогового фильтра. Пороговый фильтр разделяет участи изображения на светлые и темные. Учитывая, что OpenCV содержит ряд преобразований и обработок, была применена библиотека cvThreshold. Параметры для порогового фильтра подбираются опытным путем.

Четвертый этап – определение границ контуров. Использовалась хорошо оптимизированная функция из библиотеки cvCanny – детектор границ. Полученные контуры обводятся на исходном изображении.

Таким образом, получается список ключевых контуров, которые в дальнейшем классифицируются посредством использования заранее обученной нейросети. Если контур не относится ни к одному заранее обозначенному типу, его игнорируют.

В соответствии с изложенным алгоритмом была разработана программа испытаний. Исходный сигнальный знак приведен на рисунке 3. Результаты обработки изображения после применения порогового фильтра приведены на

рисунке 4. На рисунке 5 представлено финальное изображение сигнального знака с ключевыми контурами.

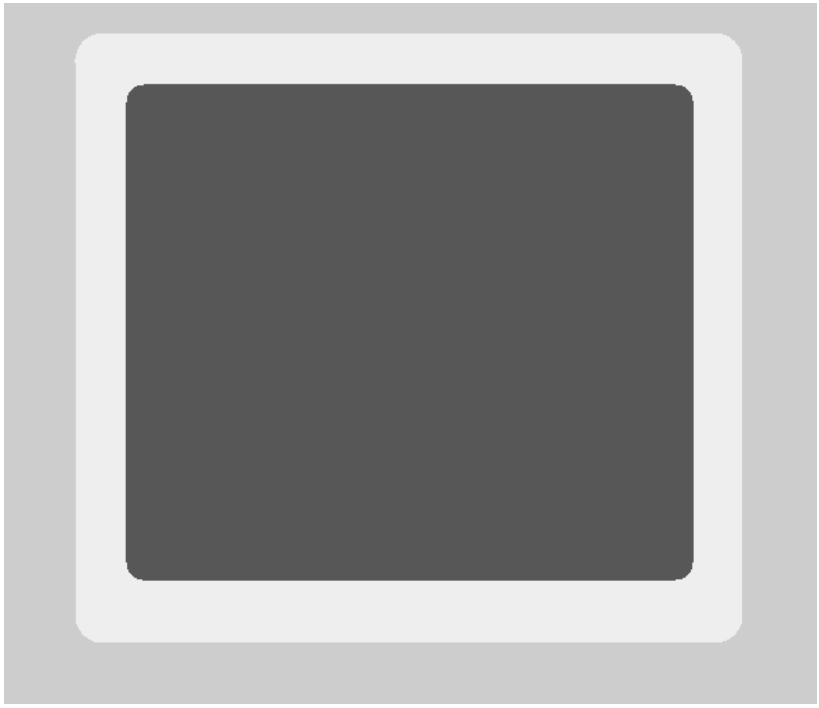

Рисунок 3. Исходный сигнальный знак

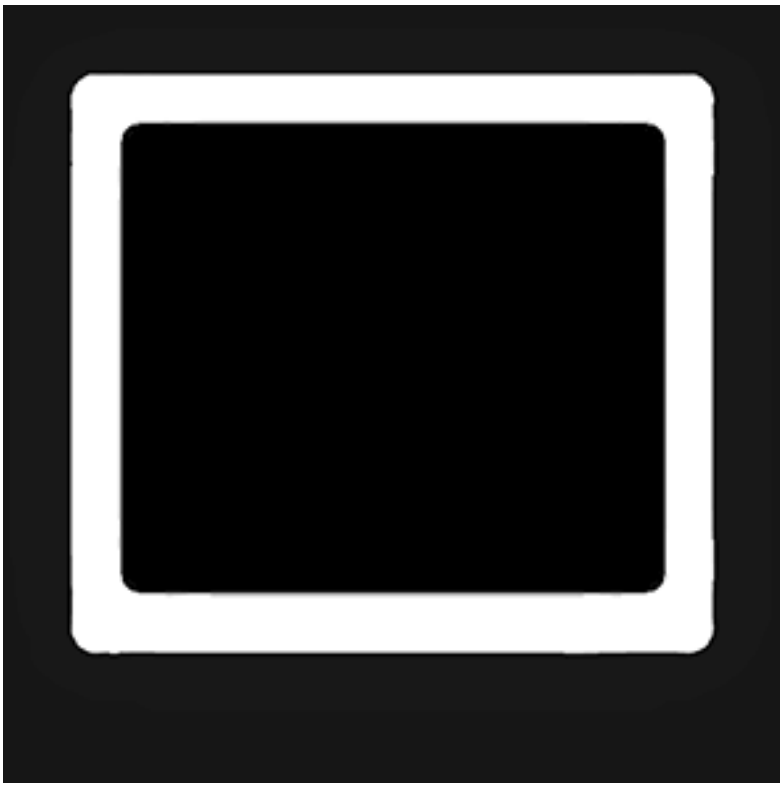

Рисунок 4. Сигнальный знак, после применения порогового фильтра

Рисунок 5. Сигнальный знак с ключевыми контурами

5.4 Создание программного кода

В ходе разработки программы были написаны исходные коды для следующих компонентов программного комплекса:

- Обработка изображения.
- Обучение нейросети.
- Алгоритм управления.

Алгоритм взаимодействия между составляющими программы следующий:

- Тестовая модель начинала движение. При движении изменялось состояние окружающей среды – появлялись новые данные.
- Данные от датчиков переходили на одноплатный компьютер, где они обрабатываются.
- В зависимости от обработки полученных данных система управления посылает сигналы управления на манипулирующие устройства.

– Изображение с камеры посылается для обработки на одноплатный компьютер. После обработки изображения, обученная нейросеть классифицирует полученные контуры и подает сигналы в систему управления.

5.5 Тестирование

При выполнении тестовых заданий алгоритм обработки сигнальных знаков начинал работать при поступлении данных об окружающей среде от датчиков. Появление объекта в обзоре камеры инициализировало процесс сопоставления полученного образа с образами, хранящимися в базе знаний.

Эксперименты были проведены на трех разных знаках различной формы (квадрат, треугольник и окружность). Изначальное положение каждого знака было по идеальному сценарию, т.е. сигнальные знаки были в прямом положении, при дневном освещении и на расстоянии в 40 см от тестовой модели (расстояние определилось в ходе проведенных экспериментов). После каждая характеристика изменялась до того момента, пока нейросеть не начинала игнорировать сигнальные знаки, т.е. не могла классифицировать полученный контур, и обратно, до идеального состояния.

Результаты работы алгоритма при распознавании сигнального знака квадратной формы приведены на рисунках 6,7,8.

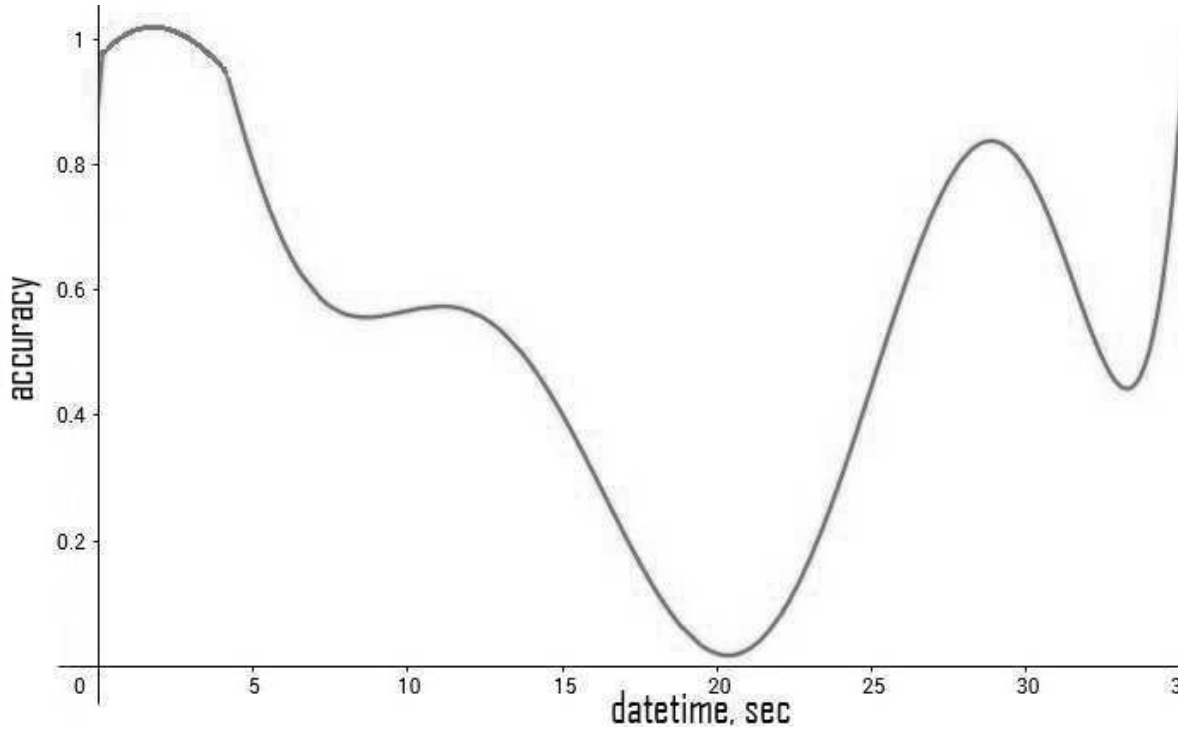

Рисунок 6. Результат работы алгоритма при распознавании сигнального знака. Расстояние до объекта.

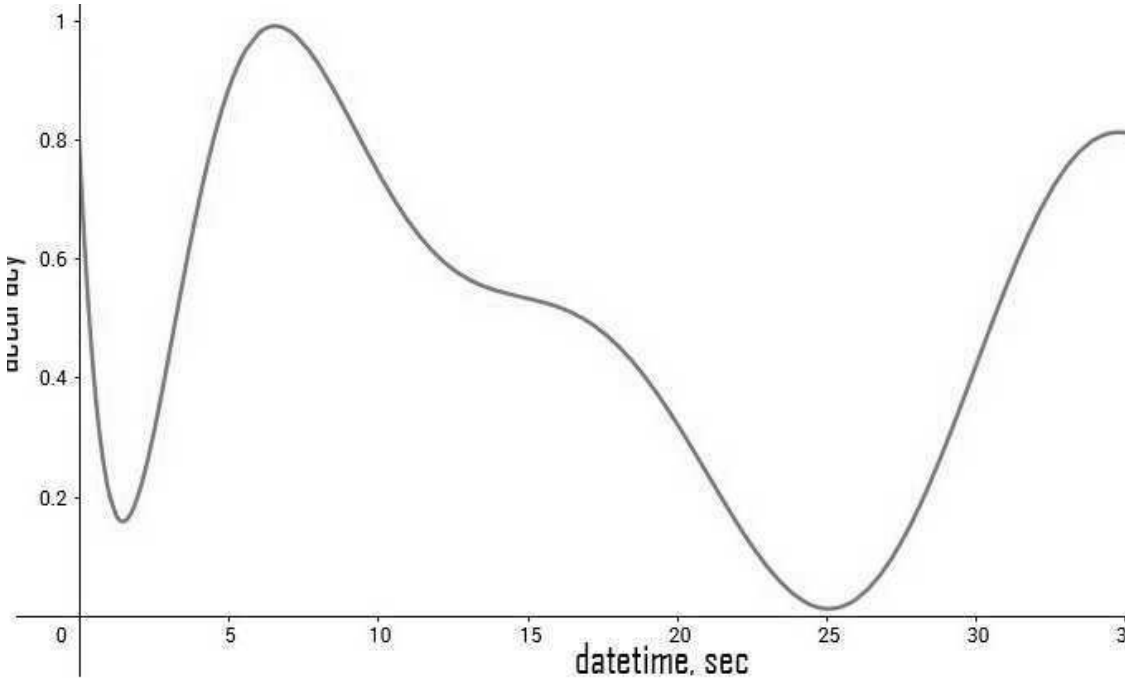

Рисунок 7. Результат работы алгоритма при распознавании сигнального знака. Угол наклона знака.

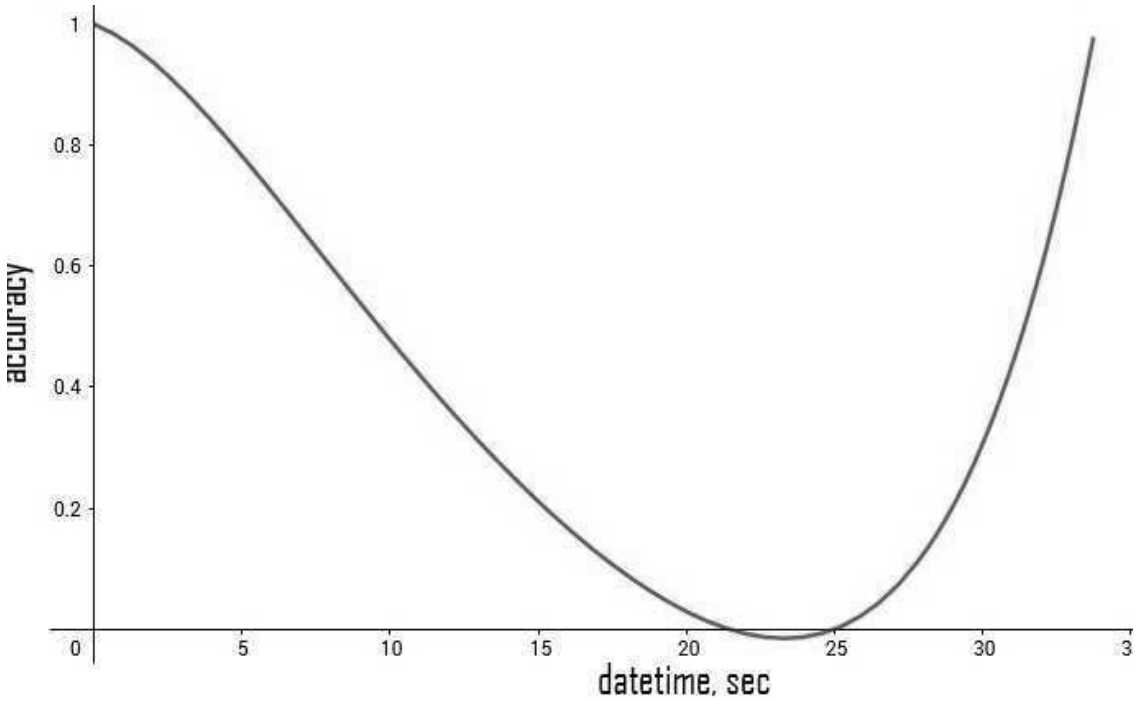

Рисунок 8. Результат работы алгоритма при распознавании сигнального знака. Освещение испытательного полигона.

ЗАКЛЮЧЕНИЕ

В результате проведенной работы был рассмотрен перечень аналогов, выявлены их общие недостатки. Изучены технологии программирования компьютерного зрения и машинного обучения. Был проведен обзор и анализ литературы по теме выпускной квалификационной работы, получена информация по современным технологиям программирования из различных источников информации.

Полученная программа является предложением решения проблемы для автомобилей, в базовой комплектации которых отсутствует система автопилотирования.

По результатам разработки можно сделать ряд выводов:

1. Результаты испытаний показали соответствие технических характеристик прототипа требованиям, предъявляемым к беспилотным транспортным средствам.
2. При идеальных условиях работы прототипа транспортного робота, т.е. хорошем освещении, прямом расположении знаков и в достаточном расстоянии до них, нейросеть достаточно точно определяет символьный знак и следует инструкции, приписанной данному знаку.
3. Поскольку идеальные условия редко достижимы, для организации движения робота по складу необходимо предусмотреть правильную расстановку сигнальных знаков, а также подсветку сигнальных знаков, например, за счет использования аварийного освещения на производственном объекте.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. International Federation of Robotics – IFR: сайт. - URL: www.ifr.org (дата обращения: 30.09.2015))

2. Белоногов А. В. Обзор систем управления приводом складских транспортных роботизированных платформ [Текст] // Технические науки в России и за рубежом: материалы VI Междунар. науч. конф. (г. Москва, ноябрь 2016 г.). — М.: Буки-Веди, 2016. — С. 11-14. — URL https://moluch.ru/conf/tech/archive/228/11326/ (дата обращения: 20.05.2018).

3. Драницкий И.О. Использование мобильных роботов для автоматизации складских помещений. // Вестник науки и образования. — 2015. — № 6 (8). С. 5-9.K. Elissa, “Title of paper if known,” unpublished.

4. Тимофеев, А. В., Юсупов, Р. М. Принципы построения интегрированных систем мульти-агентной навигации и интеллектуального управления мехатронными роботами.// International Journal "Information Technologies & Knowledge". - Vol.5, Number 3, 2011, p.273-244.

5. Черноусько, Ф.Л. Болотник, Н.Н. Градецкий, В.Г. Мобильные роботы: исследования, разработки, перспективы. – http://www.ras.ru/news/shownews.aspx?id=f5c75bcf-2fa5-40e6-b067-4492f0c5ab22&print=1].

6. M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

7. Industrial robot http://metallicheckiy-portal.ru/articles/obrabotka/roboti/promishlennie_roboti/3.

8. Transport system based on industrial robots http://helpiks.org/4-34612.html.

9. Mobile transport robothttp://poleznayamodel.ru/model/9/96834.html.

10. Orange Pi PC One platform http://www.orangepi.org/orangepione/

11. One warehouse and hundreds of robots http://roboting.ru/394-odin-sklad-i-sotni-robotov.html.

12. K. Danilyuk, “CarND Project 1: Lane Lines Detection — A Complete Pipeline “ http://towardsdatascience.com/carnd-project-1-lane-lines-detection-a-complete-pipeline-6b815037d02c.

13. A.S. Klimov, Image File Formats. Kyiv: “Diasoft Ltd.” Research and Production Company, 1995.

14. http://homepage.tudelft.nl/e3q6n/publications/1998/ICPR98LVTYPV/ICPR98LVTYPV.pdf.

15. https://www.ibm.com/support/knowledgecenter/ru/SSLVMB_24.0.0/spss/base/idh_idd_knn_variables.html.

16. Digital Picture Processing, Ed. by G. Andrews and L. Inlo, Transl. from English. Moscow: Mir, 1973. (in Russian).

17. David Puljiz, Gleb Gorbachev, Björn Hein, “mplementation of Augmented Reality in Autonomous Warehouses: Challenges and Opportunities”, http://vam-hri.xyz/files/pdf/Implementation%20of%20AR%20in%20Autonomous%20Warehouses%20-%20Challenges%20and%20Opportunities_reviewed.pdf

18. Jae-Han Park, Seung-Ho Baeg, Jaehan Koh, Kyung-Wook Park, Moon-Hong Baeg, “A new object recognition system for service robots in the smart environment “ https://ieeexplore.ieee.org/document/4407060/

19. Talgat Islamgozhayev, Maksat Kalimoldayev, Arman Eleusinov, Shokan Mazhitov, Orken Mamyrbayev, “First result in the development of a mobile robot with trajectory planning and object recognition capabilities” https://www.degruyter.com/view/j/eng.2016.6.issue-1/eng-2016-0049/eng-2016-0049.xml

20. [21] Mohammed Myasar Ali , Hui Liu , Norbert Stoll , Kerstin Thurow, “Recognition and Position Estimation for Multiple Labware Transportation Using Kinect V2 and Mobile Robots” https://www.astesj.com/publications/ASTESJ_0203154.pdf

1) Код обработка изображения

ImportImage.py

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

#reading in an image
image = mpimg.imread('test_images/challenge.jpg')

#printing out some stats and plotting
print('This image is:', type(image), 'with dimesions:', image.shape)
plt.imshow(image, cmap='gray')
# if you wanted to show a single
#color channel image called 'gray', for example, call as plt.imshow(gray, cmap='gray')

import os

from itertools import chain
from copy import deepcopy

class Lane(object):
    """
    Domain knowledge for lane recognition and lane filtering.
    We have two lanes and update them using the information we gather.
    """

    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': (255,0,0),
        '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):
    """
    Lines interpolation using a linear regression.
    Any order of polynomial can be used, but we limit ourselves with 1st order for now.
    """
    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)

    # Assisted lane lines detection for the 1st video frame
    # This can be definitely improved
    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):
    """
    Since lane line can be any order polynomial, I keep all poly coefficients
```

```
        in an array -- this is mostly for the future.
        """
        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):
    """
    Line: y = ax + b.
    A line can be described by its pair of coordinates (x1, y1), (x2, y2).
    To formalize a line, we need to compute its slope (a) and intercept (b).
    """

    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):
        """
        A simple domain logic to check whether this hough line can be a candidate
        for being a segment of a lane line.
        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):
        """
        Reference https://en.wikipedia.org/wiki/Distance_from_a_point_to_a_line
        """
        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 gimp_to_opencv_hsv(*hsv):
    """
```

```
    I use GIMP to visualize colors. This is a simple
    GIMP => CV2 HSV format converter.
    """
    return (hsv[0] / 2, hsv[1] / 100 * 255, hsv[2] / 100 * 255)

# 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)

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

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

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)

def hough_line_transform(image, rho, theta, threshold, min_line_length, max_line_gap):
    """
    A modified implementation of a suggested `hough_lines` function which allows
    Line objects initialization and in-place line filtering.
    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

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])

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.
    """

    ### Phase 1: Image Preprocessing

    hsv_image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)

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

    masked_image = draw_binary_mask(binary_mask, hsv_image)

    blank_image = np.zeros_like(image)

    edges_mask = canny(masked_image, 280, 360)

    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))

    # Snapshot 2
    out_snap2 = np.zeros_like(image)
    out_snap2 = draw_canny_edges(edges_mask, 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))

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

```
    output = draw_dashboard(output, snapshot1, snapshot2)
    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):
    """Applies the Grayscale transform
```

```
    This will return an image with only one color channel
    but NOTE: to see the returned image as grayscale
    (assuming your grayscaled image is called 'gray')
    you should call plt.imshow(gray, cmap='gray')"""
    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):
    """
    NOTE: this is the function you might want to use as a starting point once you want to
    average/extrapolate the line segments you detect to map out the full
    extent of the lane (going from the result shown in raw-lines-example.mp4
    to that shown in P1_example.mp4).

    Think about things like separating line segments by their
    slope ((y2-y1)/(x2-x1)) to decide which segments are part of the left
    line vs. the right line.  Then, you can average the position of each of
    the lines and extrapolate to the top and bottom of the lane.

    This function draws `lines` with `color` and `thickness`.
```

```
    Lines are drawn on the image inplace (mutates the image).
    If you want to make the lines semi-transparent, think about combining
    this function with the weighted_img() function below
    """
    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):
    """
    `img` should be the output of a Canny transform.

    Returns an image with hough lines drawn.
    """
    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

# Python 3 has support for cool math symbols.

def weighted_img(img, initial_img, α=0.8, β=1., λ=0.):
    """
    `img` is the output of the hough_lines(), An image with lines drawn on it.
    Should be a blank image (all black) with lines drawn on it.

    `initial_img` should be the image before any processing.

    The result image is computed as follows:

    initial_img * α + img * β + λ
    NOTE: initial_img and img must be the same shape!
    """
    return cv2.addWeighted(initial_img, α, img, β, λ)


image = mpimg.imread('test_images/challenge2.jpg')

Lane.purge()
plt.imshow(image_pipeline(image))


import os os.listdir("test_images/")

# TODO: Build your pipeline that will draw lane lines on the test_images
# then save them to the test_images directory.

VERBOSE = False
for img in os.listdir("test_images/"):
    if img.endswith('jpg'):
        Lane.purge()
        plt.figure(figsize=(12,8))
        image = mpimg.imread('test_images/{}'.format(img))
```

```
plt.imshow(image_pipeline(image))
```

Learn.py

```
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.models import load_model
from keras.layers import Dense, LSTM, Dropout
from datetime import datetime
from datetime import timedelta
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter, AutoDateLocator
import drive_callbacks # Подгон от Игорехи


# ---- Функция разделения датасета на обучающую и тестовую выборки ------------------------
# dataset    - загруженный через pandas нормализованный датасет
# test_ratio - часть тестовой выборки от общего количества данных (число от 0 до 1)
# prev_days  - количество дней, предшествующих прогнозируемому дню
def split_learn_data(batch_size, dataset, test_ratio, prev_days):
	# Проходим по всему файлу дабы выделить нужное количество обучающих и тестовых данных по всем регионам
	indices       = []
	start_position = 0
	for i in range(1, dataset.shape[0]):
		if dataset['region'][i] != dataset['region'][i - 1]:
			indices.append([start_position, int((i - start_position) * (1 - test_ratio)) + start_position, i])
			start_position = i
	# Обрабатываем край
	indices.append([start_position, int((dataset.shape[0] - start_position) * (1 - test_ratio)) + start_position, dataset.shape[0]])
	# Вычисляем количество батчей на одну эпоху
	steps_per_epoch_fit = 0
	steps_per_epoch_test = 0
	for i in range(0, len(indices)):
		# ---- Обучающие данные
		start_index   = indices[i][0] + prev_days
		stop_index    = indices[i][1]
		samples_count = stop_index - start_index
		# Определяемся с количеством батчей
		steps_per_epoch_fit = int(samples_count / batch_size)
		if samples_count % batch_size > 0:
			steps_per_epoch_fit += 1
		# ---- Тестовые данные
		start_index   = indices[i][1] + prev_days
		stop_index    = indices[i][2]
		samples_count = stop_index - start_index
		# Определяемся с количеством батчей
		steps_per_epoch_test = int(samples_count / batch_size)
		if samples_count % batch_size > 0:
			steps_per_epoch_test += 1
	return indices, steps_per_epoch_fit, steps_per_epoch_test
# ---- Функция разделения датасета по регионам - для прогонки любых данных
```

```
def split_test_data(dataset):
    # Проходим по всему файлу дабы выделить нужное количество обучающих и тестовых данных по всем регионам
    indices       = []
    start_position = 0
    for i in range(1, dataset.shape[0]):
        if dataset['region'][i] != dataset['region'][i - 1]:
            indices.append([start_position, i])
            start_position = i
    # Обрабатываем край
    indices.append([start_position, dataset.shape[0]])
    return indices
# ---- Функции генераторов ------------------------------------------------------------
# ---- Генератор обучающей выборки
# Размер входных данных: 1 + prev_days + 6 * (prev_days + 1) = 7 * (1 + prev_days)
# batch_size - количество сэмплов, скармливаемых нейронке за раз
# dataset    - загруженный через pandas нормализованный датасет
# indices    - индексы данных для обучения по каждому ФО
# prev_days  - количество дней, предшествующих прогнозируемому дню
# random       - перемешивать ли даты (индексы из отсеченной от всего датасета части обучающей выборки перемешиваются)
def generate_train_data(batch_size, dataset, indices, prev_days, random=False):
    while True:
        for i in range(0, len(indices)):
            start_index   = indices[i][0] + prev_days
            stop_index    = indices[i][1]
            samples_count = stop_index - start_index
            # Определяемся с количеством батчей
            batches = int(samples_count / batch_size)
            if samples_count % batch_size > 0:
                batches += 1
            # Определяемся с порядком следования батчей
            batch_sequence = np.arange(0, batches)
            if random == True:
                np.random.shuffle(batch_sequence)
            # Генерируем батчи для данного ФО
            for j in batch_sequence:
                interval = None
                if j == batches - 1:
                    interval = np.arange(j * batch_size + start_index, stop_index)
                else:
                    interval = np.arange(j * batch_size + start_index, (j + 1) * batch_size + start_index)
                # Проверяем, необходимо ли перемешивать данные
                if random == True:
                    np.random.shuffle(interval)
                # Заполняем батчи, определившись с индексами
                x_train = []
                y_train = []
                for k in interval:
                    # ---- ВХОДНЫЕ ДАННЫЕ
                    sample = []
                    # Добавляем данные по региону
                    sample.append(dataset['region'][k])
```

```
                                        # Добавляем данные по потреблению, предшевствующие
прогнозу
                                        for l in range(1, prev_days + 1):
                                                sample.append(dataset['consumption'][k - l])
                                        # Данные о длине светового дня
                                        for l in range(0, prev_days + 1):
                                                sample.append(dataset['daylights'][k - l])
                                        # Данные о праздниках/выходных
                                        for l in range(0, prev_days + 1):
                                                sample.append(dataset['holiday'][k - l])
                                        # Данные о дате
                                        for l in range(0, prev_days + 1):
                                                sample.append(dataset['date'][k - l])
                                        # Данные о дне недели
                                        for l in range(0, prev_days + 1):
                                                sample.append(dataset['day'][k - l])
                                        # Данные о температуре
                                        for l in range(0, prev_days + 1):
                                                sample.append(dataset['air_temp'][k - l])
                                        # Данные об облачности
                                        for l in range(0, prev_days + 1):
                                                sample.append(dataset['sky'][k - l])
                                        # Добавляем в батч
                                        x_train.append([sample])
                                        # ---- ВЫХОДНЫЕ ДАННЫЕ
                                        y_train.append(dataset['consumption'][k])
                                # Передаем нейронке
                                yield (np.array(x_train), np.array(y_train))
# ---- Генератор тестовой выборки
def generate_test_data(batch_size, dataset, indices, prev_days, random=False):
        while True:
                for i in range(0, len(indices)):
                        start_index   = indices[i][1] + prev_days
                        stop_index    = indices[i][2]
                        samples_count = stop_index - start_index
                        # Определяемся с количеством батчей
                        batches = int(samples_count / batch_size)
                        if samples_count % batch_size > 0:
                                batches += 1
                        # Определяемся с порядком следования батчей
                        batch_sequence = np.arange(0, batches)
                        if random == True:
                                np.random.shuffle(batch_sequence)
                        # Генерируем батчи для данного ФО
                        for j in batch_sequence:
                                interval = None
                                if j == batches - 1:
                                        interval = np.arange(j * batch_size + start_index, stop_index)
                                else:
                                        interval = np.arange(j * batch_size + start_index, (j + 1) *
batch_size + start_index)
                                # Смотрим - необходимо ли перемешивать данные
                                if random == True:
                                        np.random.shuffle(interval)
```

```
                # Заполняем батчи, определившись с индексами
                x_test = []
                y_test = []
                for k in interval:
                        # ---- ВХОДНЫЕ ДАННЫЕ
                        sample = []

                        sample.append(dataset['r
                        for l in range(1, prev_days + 1):
                                sample.append(dataset['consumption'][k - l])
                        for l in range(0, prev_days + 1):
                                sample.append(dataset['daylights'][k - l])
                        for l in range(0, prev_days + 1):
                                sample.append(dataset['holiday'][k - l])
                        for l in range(0, prev_days + 1):
                                sample.append(dataset['date'][k - l])
                        for l in range(0, prev_days + 1):
                                sample.append(dataset['day'][k - l])
                        for l in range(0, prev_days + 1):
                                sample.append(dataset['air_temp'][k - l])
                        for l in range(0, prev_days + 1):
                                sample.append(dataset['sky'][k - l])
                        # Добавляем в батч
                        x_test.append([sample])
                        # ---- ВЫХОДНЫЕ ДАННЫЕ
                        y_test.append(dataset['consumption'][k])
                # Передаем нейронке
                yield (np.array(x_test), np.array(y_test))


# ---- Функция обучения нейронной сети ----------------------------------------------------
def learnFunction(dataset_path, save_model_path, fit_plot_path, batch_size, epochs, test_ratio,
prev_days, random=False):
        # ---- ПОДГОТОВКА ДАННЫХ ------------------------------------------------------------
        # Загружаем датасет
        dataset = pd.read_csv(dataset_path, sep=';')
        # Разделяем данные на обучающие и тестовые
        indices, steps_per_epoch_fit, steps_per_epoch_test = split_learn_data(batch_size, dataset,
test_ratio, prev_days)
        # Выводим данные
        print('batch_size:          {0}'.format(batch_size))
        print('epochs:              {0}'.format(epochs))
        print('steps_per_epoch_fit:  {0}'.format(steps_per_epoch_fit))
        print('steps_per_epoch_test: {0}'.format(steps_per_epoch_test))
        print('')

        # ---- ИНИЦИАЛИЗАЦИЯ АРХИТЕКТУРЫ СЕТИ ----------------------------------------
-------
        model = Sequential()
        # Добавляю слои
        model.add(LSTM(62, input_shape=(1, 7 * (1 + prev_days))))       # Было 62
        #model.add(Dropout(0.15))  # Предотращение переобучения
        model.add(Dense(1))
        model.compile(loss='mean_squared_error', optimizer='Adam', metrics=['accuracy'])
```

```
        # ---- ОБУЧЕНИЕ ----------------------------------------------------------------------
        # Инициализация экземпляров генераторов
        train_data_generator = generate_train_data(batch_size, dataset, indices, prev_days,
random=random)
        test_data_generator = generate_test_data(batch_size, dataset, indices, prev_days,
random=random)
        # Игорехин подгон
        statistic = drive_callbacks.Statistic()
        # Обучение
        history = model.fit_generator(train_data_generator, steps_per_epoch=steps_per_epoch_fit,
epochs=epochs,
                                      validation_data=test_data_generator,
validation_steps=steps_per_epoch_test,
                                      callbacks=[statistic])
        # Сохраняем модель для дальнейшего использования
        model.save(save_model_path)

        # ---- ВИЗУАЛИЗАЦИЯ ------------------------------------------------------------------
        # Отрисовываем графики MSE и accuracy
        visualizationFitModel(history, fit_plot_path)


# ---- Тестирование ------------------------------------------------------------------
# ---- Вспомогательная функция нормализации данных на-лету
# dataset                   - датасет исходных не нормализованных данных
# region_max, region_min, region_code - значения нормализации по ФО
# max_temp, min_temp              - значения нормализации температуры
# prev_days                   - количество дней, предшествующих прогнозируемому дню
# index                     - позиция в файле
def normalizationData(dataset, region_max, region_min, region_code, max_temp, min_temp,
prev_days, index):
        # ---- ВХОДНЫЕ ДАННЫЕ
        sample = []
        # Инициализация минимального/максимального значения нормализации
        max_value = region_max[dataset['region'][index]]
        min_value = region_min[dataset['region'][index]]
        # Добавляем данные по региону
        sample.append( region_code[dataset['region'][index]] )
        # Добавляем данные по потреблению, предшевствующие прогнозу
        for i in range(1, prev_days + 1):
                sample.append( (dataset['consumption'][index - i] - min_value) / (max_value -
min_value) )
        # Данные о длине светового дня
        for i in range(0, prev_days + 1):
                sample.append( dataset['daylights'][index - i] / 1440.0 )
        # Данные о праздниках/выходных
        for i in range(0, prev_days + 1):
                sample.append(dataset['holiday'][index - i])
        # Данные о дате
        for i in range(0, prev_days + 1):
                current_dt = datetime( int(dataset['date'][index - i][6:10]), int(dataset['date'][index -
i][3:5]), int(dataset['date'][index - i][0:2]) )
                start_dt   = datetime( int(current_dt.year), 1, 1)
```

```
                sample.append( int((current_dt - start_dt).days) / 366.0 )
        # Данные о дне недели
        for i in range(0, prev_days + 1):
                sample.append( datetime(int(dataset['date'][index - i][6:10]),
int(dataset['date'][index - i][3:5]), int(dataset['date'][index - i][0:2])).weekday() / 6.0 )
        # Данные о температуре
        for i in range(0, prev_days + 1):
                sample.append( (dataset['air_temp'][index - i] - min_temp) / (max_temp -
min_temp) )
        # Данные об облачности
        for i in range(0, prev_days + 1):
                sample.append( dataset['sky'][index - i] / 9.0 )
        # Добавляем в батч
        input_data = np.array([[sample]])
        # ---- ВЫХОДНЫЕ ДАННЫЕ
        output_data = (dataset['consumption'][index] - min_value) / (max_value - min_value)
        # Возвращаем данные
        return input_data, output_data
# ---- Вспомогательная функция денормализации результата
def denormalizationResult(value, max_value, min_value):
        return (value * (max_value - min_value) + min_value)
# ---- Вычисление метрик
def metrics(source_values, forecast_values, region, filepath):
        # Массивы результатов
        region_list = []
        MAE_list    = []
        MAPE_list   = []
        max_AE_list = []
        min_AE_list = []

        # Вычисление метрик
        for reg in region:
                index = region[reg]
                region_list.append(reg)
                # Преобразуем данные
                s_data = np.array(source_values[index])
                f_data = np.array(forecast_values[index])
                # Вычисления
                MAE_list.append('{0:.1f}'.format(np.mean(np.absolute(s_data - f_data))))
                max_AE_list.append('{0:.1f}'.format(np.max(np.absolute(s_data - f_data))))
                min_AE_list.append('{0:.1f}'.format(np.min(np.absolute(s_data - f_data))))
                MAPE_list.append('{0:.2f}'.format(np.mean(np.absolute(s_data - f_data) / f_data)
* 100.0))

        # Сохранение на диск
        result = pd.DataFrame({ 'region': region_list, 'MAE, MW*h': MAE_list, 'maxAE, MW*h':
max_AE_list, 'minAE, MW*h': min_AE_list, 'MAPE, %': MAPE_list })
        result.to_csv(filepath, sep=';')
# ---- Прогоняем валидационную выборку, строим графики, оцениваем метрики
def validationFunction(dataset_path, normalization_data_path, model_path, result_dir, batch_size,
test_ratio, prev_days, max_temp, min_temp):
        # ---- ПЕРЕМЕННЫЕ ------------------------------------------------------------------
        region          = {'ural': 0, 'eastern': 1, 'northwestern': 2, 'siberia': 3, 'volga': 4, 'south': 5,
'central': 6}
```

```
    datetime_list  = [[], [], [], [], [], [], []]
    source_value   = [[], [], [], [], [], [], []]
    forecast_value = [[], [], [], [], [], [], []]
    absolute_error = [[], [], [], [], [], [], []]


    # ---- ПОДГОТОВКА ------------------------------------------------------------------
    # Загружаем параметры нормализации
    parametrs = pd.read_csv(normalization_data_path, sep=';')
    # Создаем необходимые для нормализации словари
    region_code = {}
    region_max = {}                    # Максимальное значение потребления для ФО
    region_min = {}                    # Минимальное значение потребления для ФО
    for i in range(0, parametrs.shape[0]):
        region_code[parametrs['region'][i]] = parametrs['region_code'][i] / 6.0
        region_max[parametrs['region'][i]]  = parametrs['region_max'][i] + 0.05 *
parametrs['region_max'][i]
        region_min[parametrs['region'][i]]  = parametrs['region_min'][i] - 0.05 *
parametrs['region_min'][i]
    # Загружаем датасет нормализованных данных
    dataset = pd.read_csv(dataset_path, sep=';')
    # Разделяем данные на обучающие и тестовые
    indices, steps_per_epoch_fit, steps_per_epoch_test = split_learn_data(batch_size, dataset,
test_ratio, prev_days)
    # Загружаем модель
    model = load_model(model_path)


    # ---- ТЕСТОВЫЙ ПРОГОН -------------------------------------------------------------
    for i in range(0, len(indices)):
        start_index  = indices[i][1] + prev_days
        stop_index   = indices[i][2]
        for j in range(indices[i][1] + prev_days, indices[i][2]):
            x_data, y_data = normalizationData(dataset, region_max, region_min,
region_code, max_temp, min_temp, prev_days, j)
            # Загружаем в нейронку (на выходе имеем - [[0.3313307]])
            y_value = model.predict(x_data)
            # Проводим денормализацию
            y_value = denormalizationResult(y_value[0][0],
region_max[dataset['region'][j]], region_min[dataset['region'][j]])
            # Добавляем в датасет результатов тестирования
            # Определяем индекс "нужного" подмассива
            region_index = region[dataset['region'][j]]
            datetime_list[region_index].append(datetime(int(dataset['date'][j][6:10]),
int(dataset['date'][j][3:5]), int(dataset['date'][j][0:2])))
            source_value[region_index].append(dataset['consumption'][j])
            forecast_value[region_index].append(y_value)
            absolute_error[region_index].append(abs(y_value -
dataset['consumption'][j]))

    # ---- ВИЗУАЛИЗАЦИЯ РЕЗУЛЬТАТОВ ----------------------------------------------------
    for reg in region:
        index = region[reg]
```

```
            visualizationTest(datetime_list[index], source_value[index], forecast_value[index], absolute_error[index], reg, result_dir + 'validation\\{0}.jpg'.format(reg))

        # ---- РАСЧЕТ МЕТРИК ----------------------------------------------------------------
        metrics(source_value, forecast_value, region, result_dir + 'error - validation.csv')
# ---- Прогоняем произвольную выборку, строим графики, оцениваем метрики
def testFunction(dataset_path, normalization_data_path, model_path, result_dir, prev_days, max_temp, min_temp):
        # ---- ПЕРЕМЕННЫЕ -------------------------------------------------------------------
        region          = {'ural': 0, 'eastern': 1, 'northwestern': 2, 'siberia': 3, 'volga': 4, 'south': 5, 'central': 6}
        datetime_list  = [[], [], [], [], [], [], []]
        source_value   = [[], [], [], [], [], [], []]
        forecast_value = [[], [], [], [], [], [], []]
        absolute_error = [[], [], [], [], [], [], []]


        # ---- ПОДГОТОВКА -------------------------------------------------------------------
        # Загружаем параметры нормализации
        parametrs = pd.read_csv(normalization_data_path, sep=';')
        # Создаем необходимые для нормализации словари
        region_code = {}
        region_max = {}                 # Максимальное значение потребления для ФО
        region_min = {}                 # Минимальное значение потребления для ФО
        for i in range(0, parametrs.shape[0]):
                region_code[parametrs['region'][i]] = parametrs['region_code'][i] / 6.0
                region_max[parametrs['region'][i]]  = parametrs['region_max'][i] + 0.05 * parametrs['region_max'][i]
                region_min[parametrs['region'][i]]  = parametrs['region_min'][i] - 0.05 * parametrs['region_min'][i]
        # Загружаем датасет нормализованных данных
        dataset = pd.read_csv(dataset_path, sep=';')
        indices = split_test_data(dataset)
        # Загружаем модель
        model = load_model(model_path)


        # ---- ТЕСТОВЫЙ ПРОГОН --------------------------------------------------------------
        for i in range(0, len(indices)):
                start_index   = indices[i][0] + prev_days
                stop_index    = indices[i][1]
                for j in range(indices[i][0] + prev_days, indices[i][1]):
                        x_data, y_data = normalizationData(dataset, region_max, region_min, region_code, max_temp, min_temp, prev_days, j)
                        # Загружаем в нейронку (на выходе имеем - [[0.3313307]])
                        y_value = model.predict(x_data)
                        # Проводим денормализацию
                        y_value = denormalizationResult(y_value[0][0], region_max[dataset['region'][j]], region_min[dataset['region'][j]])
                        # Добавляем в датасет результатов тестирования
                        # Определяем индекс "нужного" подмассива
                        region_index = region[dataset['region'][j]]
                        datetime_list[region_index].append(datetime(int(dataset['date'][j][6:10]), int(dataset['date'][j][3:5]), int(dataset['date'][j][0:2])))
```

```
                        source_value[region_index].append(dataset['consumption'][j])
                        forecast_value[region_index].append(y_value)
                        absolute_error[region_index].append(abs(y_value -
dataset['consumption'][j]))

        # ---- ВИЗУАЛИЗАЦИЯ РЕЗУЛЬТАТОВ ----------------------------------------------------
        for reg in region:
                index = region[reg]
                visualizationTest(datetime_list[index], source_value[index], forecast_value[index],
absolute_error[index], reg, result_dir + 'test\\{0}.jpg'.format(reg))

        # ---- РАСЧЕТ МЕТРИК ----------------------------------------------------------------
        metrics(source_value, forecast_value, region, result_dir + 'error - test.csv')


# ---- Визуализация ------------------------------------------------------------------
# ---- Отрисовка графиков результатов
def visualizationTest(datetime_values, source_values, forecast_values, error_values, region,
filePath):
        plt.figure(1, figsize=(20, 11.25), dpi=96)
        # ---- Отрисовка значений
        ax = plt.subplot(211)
        plt.title('{0}'.format(region))
        plt.xlabel('datetime')
        plt.ylabel('consumption, MW*h')
        plt.plot(datetime_values, source_values, 'g', label='input')
        plt.plot(datetime_values, forecast_values, 'b', label='forecast')
        plt.legend()
        plt.grid(True)
        ax.xaxis.set_major_locator(AutoDateLocator())
        ax.xaxis.set_major_formatter(DateFormatter('%d.%m.%Y'))
        plt.setp(plt.gca().get_xticklabels(), rotation = 0,horizontalalignment = 'center')
        # ---- Отрисовка погрешности
        ax = plt.subplot(212)
        plt.xlabel('datetime')
        plt.ylabel('absolute error, MW*h')
        plt.plot(datetime_values, error_values, 'r', label='error')
        plt.grid(True)
        ax.xaxis.set_major_locator(AutoDateLocator())
        ax.xaxis.set_major_formatter(DateFormatter('%d.%m.%Y'))
        plt.setp(plt.gca().get_xticklabels(), rotation = 0,horizontalalignment = 'center')
        # ---- Сохраняем
        plt.savefig(filePath)
        plt.close('all')
# ---- Отрисовка метрик обучения модели
def visualizationFitModel(history, filePath):
        plt.figure(1, figsize=(20, 11.25), dpi=96)
        # ---- Отрисока среднеквадратичной ошибки
        ax = plt.subplot(211)
        plt.title('MSE')
        plt.plot(history.history['loss'], color='orangered')
        plt.plot(history.history['val_loss'], color='dodgerblue')
        plt.ylabel('MSE')
        plt.xlabel('epoch')
```

```
        plt.grid(True)
        plt.legend(['train', 'validation'], loc='upper left')
        # ---- Отрисовка точности
        ax = plt.subplot(212)
        plt.title('Accuracy')
        plt.plot(1 - np.sqrt(history.history['loss']), color='orangered')
        plt.plot(1 - np.sqrt(history.history['val_loss']), color='dodgerblue')
        plt.ylabel('accuracy')
        plt.xlabel('epoch')
        plt.grid(True)
        plt.legend(['train', 'validation'], loc='upper left')
        # ---- Сохраняем
        plt.savefig(filePath)
        plt.close('all')


if __name__ == "__main__":
        # ---- КОНСТАНТЫ ----------------------------------------------------------------------
        max_temp   = 50                         # Значения используемые в генераторах...
        min_temp   = -70                #       ...при нормализации температуры
        prev_days  = 3
        test_ratio = 0.15               # Часть валидационной выборки
        batch_size = 32
        epochs     = 88 # Было 88
        random     = True               # Перемешивание входных данных
        # ---- ПУТИ ---------------------------------------------------------------------------
        # Обучение
        NORM_DATASET_PATH                                                       =
'D:\\Bitbucket\\neural_forecast\\final_dataset\\dataset_normalization.csv'
        SAVE_MODEL_PATH   = 'D:\\Bitbucket\\neural_forecast\\model\\model.h5'
        FIT_RESULT_PATH   = 'D:\\Bitbucket\\neural_forecast\\result\\fit.jpg'
        # Валидация
        SRC_DATASET_PATH                                                        =
'D:\\Bitbucket\\neural_forecast\\final_dataset\\dataset_source.csv'
        NORMALIZATION_DATA_PATH                                                 =
'D:\\Bitbucket\\neural_forecast\\final_dataset\\normalization.csv'
        MODEL_PATH            = 'D:\\Bitbucket\\neural_forecast\\model\\model.h5'
        TEST_RESULT_DIR       = 'D:\\Bitbucket\\neural_forecast\\result\\'

        # Проводим обучение
        #learnFunction(NORM_DATASET_PATH,                SAVE_MODEL_PATH,
FIT_RESULT_PATH, batch_size, epochs, test_ratio, prev_days, random=random)
        # Проводим тестирование на валидационной выборке
        validationFunction(SRC_DATASET_PATH,          NORMALIZATION_DATA_PATH,
MODEL_PATH, TEST_RESULT_DIR, batch_size, test_ratio, prev_days, max_temp, min_temp)
        # Прогоняем произвольную выборку
        testFunction('D:\\Bitbucket\\neural_forecast\\test_dataset\\dataset\\dataset_source.csv',
NORMALIZATION_DATA_PATH,    MODEL_PATH,    TEST_RESULT_DIR,    prev_days,
max_temp, min_temp)
```

Drive_callback.py

```
from keras.callbacks import Callback



class ValAccMonitor(Callback):
    '''
    Stop training when a validation accuracy has reached given value
    '''

    def __init__(self, acc_value=0.9):

            self.acc_value = acc_value


    def on_epoch_end(self, epoch, logs=None):

            current_val = logs.get('val_acc')

            if current_val >= self.acc_value:
                    print('Early stopping')
                    print('Accuracy has reached given value')
                    self.model.stop_training = True


class Statistic(Callback):
    '''
    Monitor metrics (train_loss, train_acc, val_loss, val_acc) for each epoch
    and train_loss, train_acc for each batch
    '''

    def __init__(self):

            self.stat = {}
            self.stat['acc'] = []
            self.stat['loss'] = []
            self.stat['val_acc'] = []
            self.stat['val_loss'] = []
            self.stat['batch_acc'] = []
            self.stat['batch_loss'] = []


    def on_epoch_end(self, epoch, logs = None):

            self.stat['loss'].append(logs.get('loss'))
            self.stat['acc'].append(logs.get('acc'))
            self.stat['val_loss'].append(logs.get('val_loss'))
            self.stat['val_acc'].append(logs.get('val_acc'))


    def on_batch_end(self, batch, logs = None):
```

```
            self.stat['batch_loss'].append(logs.get('loss'))
            self.stat['batch_acc'].append(logs.get('acc'))


class SaveBest(Callback):
    '''
    Save model has showed the best result for validation loss
    '''

    def __init__(self, filename, value=0.8, min_epoch=3):

            self.filename = filename
            self.best = value
            self.min_epoch = min_epoch


    def on_epoch_end(self, epoch, logs=None):

            if epoch > self.min_epoch:

                    cur = logs.get('val_loss')

                    if cur < self.best:

                            self.best = cur
                            self.model.save(self.filename)
```

Final_dataset.py

```
import os
import pandas as pd
from datetime import datetime


# ---- ПУТИ ---------------------------------------------------------------------------
#CONSUMPTION_DIR = 'D:\\Bitbucket\\neural_forecast\\consumption_days\\'
#DAYLIGHTS_DIR   = 'D:\\Bitbucket\\neural_forecast\\daylight_hours\\'
#HOLIDAYS_DIR    = 'D:\\Bitbucket\\neural_forecast\\holidays\\'
#WEATHER_DIR     = 'D:\\Bitbucket\\neural_forecast\\noaa_weather_data\\'
#SAVE_DIR        = 'D:\\Bitbucket\\neural_forecast\\final_dataset\\'
CONSUMPTION_DIR = 'D:\\Bitbucket\\neural_forecast\\test_dataset\\consumption_days\\'
DAYLIGHTS_DIR   = 'D:\\Bitbucket\\neural_forecast\\test_dataset\\daylight_hours\\'
HOLIDAYS_DIR    = 'D:\\Bitbucket\\neural_forecast\\test_dataset\\holidays\\'
WEATHER_DIR     = 'D:\\Bitbucket\\neural_forecast\\test_dataset\\noaa_weather_data\\'
SAVE_DIR        = 'D:\\Bitbucket\\neural_forecast\\test_dataset\\dataset\\'
# ---- ПЕРЕМЕННЫЕ ---------------------------------------------------------------------
region = ['ural', 'eastern', 'northwestern', 'siberia', 'volga', 'south', 'central']
# Результирующие массивы
datetime_list      = []
day_of_week_list   = []
region_list        = []
consumption_list   = []
daylights_list     = []
holidays_list      = []
air_temp_list      = []
sky_condition_list = []

# Загружаем данные о выходных и праздниках (одинаковые данные для всех ФО)
holidays_df = pd.read_csv(HOLIDAYS_DIR + 'holidays_final.csv', sep=';')
# Перебираем регионы
for reg in region:
    print('Proccessing {0}'.format(reg))
    # ---- ЗАГРУЖАЕМ ВСЕ НЕОБХОДИМЫЕ ДАТАСЕТЫ ----------------------------------
----------
    consumption_df = pd.read_csv(CONSUMPTION_DIR + '{0}.csv'.format(reg), sep=';')
    daylights_df   = pd.read_csv(DAYLIGHTS_DIR + '{0}.csv'.format(reg), sep=';')
    weather_df     = pd.read_csv(WEATHER_DIR + '{0}.csv'.format(reg), sep=';')

    # ---- ЗАПОЛНЯЕМ ДАННЫМИ ---------------------------------------------------------
    # Заполняем результирующие массивы данными из датасета посуточного
потребления
    for i in range(0, consumption_df.shape[0]):
        # ---- ОСНОВНЫЕ ДАННЫЕ
        datetime_list.append(consumption_df['date'][i])
        consumption_list.append(consumption_df['consumption'][i])
        region_list.append(reg)
        day_of_week_list.append(datetime(int(consumption_df['date'][i][6:10]),
int(consumption_df['date'][i][3:5]), int(consumption_df['date'][i][0:2])).weekday())

        success_flag = 0x00
```

```
                for j in range(0, daylights_df.shape[0]):
                        if int(daylights_df['date'][j][0:2]) == int(consumption_df['date'][i][0:2]) and
int(daylights_df['date'][j][3:5]) == int(consumption_df['date'][i][3:5]):

        daylights_list.append('{0:.1f}'.format(daylights_df['daylights_min'][j]))
                                success_flag = 0x01
                                break
                # Проверяем успех выполнения
                if success_flag == 0x00:
                        daylights_list.append(-9999)

                # ---- ВЫХОДНЫЕ И ПРАЗДНИЧНЫЕ ДНИ
                success_flag = 0x00
                for j in range(i, holidays_df.shape[0]):
                        if holidays_df['date'][j] == consumption_df['date'][i]:
                                holidays_list.append('{0:.1f}'.format(holidays_df['id'][j]))
                                success_flag = 0x01
                                break
                if success_flag == 0x00:
                        holidays_list.append(-9999)

                success_flag = 0x00
                for j in range(i, weather_df.shape[0]):
                        if weather_df['datetime'][j] == consumption_df['date'][i]:
                                air_temp_list.append('{0:.1f}'.format(weather_df['air_temp'][j]))
                                sky_condition_list.append(weather_df['sky'][j])
                                success_flag = 0x01
                                break
                if success_flag == 0x00:
                        air_temp_list.append(-9999)
                        sky_condition_list.append(-9999)

    # ---- СОХРАНЯЕМ ДАТАСЕТ ------------------------------------------------------------------
    dataset = pd.DataFrame({ 'date': datetime_list, 'day': day_of_week_list, 'consumption':
consumption_list, 'region': region_list, 'daylights': daylights_list, 'holiday': holidays_list,
'air_temp': air_temp_list, 'sky': sky_condition_list })
    dataset.to_csv(SAVE_DIR + 'dataset_source.csv', sep=';')
```

Command_opt.cpp

```
#define __OPTIONS_H
#include <algorithm>
#include <string>
#include <vector>
#include <sstream>
#include <iomanip>
#include <iostream>
#include <iterator>
#include <stdio.h>


using namespace std;


#define MAX_STRING_SIZE   0x100
#define TAB_SIZE          20




static const char *typeNames[]={"int","float","double","string","bool"};


const char* getTypename(int& val)       {return typeNames[0];}
const char* getTypename(float& val)     {return typeNames[1];}
const char* getTypename(double& val)            {return typeNames[2];}
const char* getTypename(string& val)    {return typeNames[3];}
const char* getTypename(bool& val)      {return typeNames[4];}


class COptions
{
  public:


    COptions(int pn,char*ps[]) : m_exename(ps[0])
    {
            for(int i=1;i<pn;i++)
            {
                    m_strings.push_back(string(ps[i]));
            }
    }

    template<class T> void Parse(const char* optString,T & optVar,const char*comment,T
defaultval)
    {
            //~ char HelpString[MAX_STRING_SIZE];
            //~ for(int i=0;i<MAX_STRING_SIZE;i++)
            //~ {
                    //~ HelpString[i] = 0;
            //~ }
            //~ memset(HelpString,0,MAX_STRING_SIZE);
```

```
		ostringstream ss;

		ss.setf(ios::left);

		ss	<<setw(TAB_SIZE)<<optString
			<<setw(TAB_SIZE)<<getTypename(optVar)
			<<setw(TAB_SIZE)<<comment
			<<setw(TAB_SIZE)<<defaultval;

		m_help.push_back(ss.str());

		vector<string>::iterator current=
find(m_strings.begin(),m_strings.end(),string(optString));

		optVar =  defaultval;



		if(current!=m_strings.end())
		{
			if(current+1==m_strings.end())
				{
					m_error.push_back(string("\nUncomplete parm error:"));
					m_error.push_back(*current);
					m_strings.erase(current,current+1);
					m_error.push_back(string("\n"));
					return ;
				}
			if(!Read(*(current+1),optVar))
			{
				m_error.push_back(string("\nRead argument error:"));
				m_error.push_back(*current);
				m_error.push_back(*(current+1));
				m_error.push_back(string("\n"));
			}
			m_strings.erase(current,current+2);
		}

	}

	template<class T> void Parse(const char* optString,T & optVar,const char*comment)
	{
		//~ char HelpString[MAX_STRING_SIZE];
		//~ for(int i=0;i<MAX_STRING_SIZE;i++)
		//~ {
			//~ HelpString[i] = 0;
		//~ }

		//~ memset(HelpString,0,MAX_STRING_SIZE);

		ostringstream ss;

		ss.setf(ios::left);
```

```
		ss	<<setw(TAB_SIZE)<<optString
			<<setw(TAB_SIZE)<<getTypename(optVar)
			<<setw(TAB_SIZE)<<comment
			<<setw(TAB_SIZE)<<"No default";

		m_help.push_back(ss.str());

		vector<string>::iterator current=
find(m_strings.begin(),m_strings.end(),string(optString));

		//optVar =  defaultval;



		if(current!=m_strings.end())
		{
			if(current+1==m_strings.end())
				{
					m_error.push_back(string("\nUncomplete parm error:"));
					m_error.push_back(*current);
					m_strings.erase(current,current+1);
					m_error.push_back(string("\n"));
					return ;
				}
			if(!Read(*(current+1),optVar))
			{
				m_error.push_back(string("\nRead argument error:"));
				m_error.push_back(*current);
				m_error.push_back(*(current+1));
				m_error.push_back(string("\n"));
			}
			m_strings.erase(current,current+2);
		}
		else
		{
			m_error.push_back(string("\nParameter must be defined:"));
			m_error.push_back(string(optString));
			m_error.push_back(string("\n"));

		}

	}

	bool IsOk()
	{
		return !(m_strings.size()||m_error.size());
	}

	void Usage(const char* Mess)
	{
		cerr<<"\n"<<Mess<<"\n";
		if(m_strings.size())
		{
```

```
            cerr<<"Unknow Parametr(s) : ";
            copy(m_strings.begin(),m_strings.end(),ostream_iterator<string>(cerr," "));
            cerr<<"\n";
        }
            if(m_error.size())
            {
            copy(m_error.begin(),m_error.end(),ostream_iterator<string>(cerr," "));
            cerr<<"\n";
        }

            cerr<<"\nUsage : ";
            cerr<<m_exename<<"\n";

            cerr.setf(ios::left);

            cerr    <<setw(TAB_SIZE)<<"Option String"
                    <<setw(TAB_SIZE)<<"Type Of Argument"
                    <<setw(TAB_SIZE)<<"Comment"
                    <<setw(TAB_SIZE)<<"Default Value"
                    <<"\n\n";

            copy(m_help.begin(),m_help.end(),ostream_iterator<string>(cerr," \n"));

  };

private:

  vector<string> m_strings;
    string              m_exename;
  vector<string> m_help;
  vector<string> m_error;
private:


  bool Read(string& str,float&optVar)
  {
          sscanf(str.c_str(),"%f",&optVar);
          return true;
  }
  bool Read(string& str,double&optVar)
  {
          sscanf(str.c_str(),"%lf",&optVar);
          return true;
  }
  bool Read(string& str,int&optVar)
  {
          sscanf(str.c_str(),"%d",&optVar);
          return true;
  }
  bool Read(string& str,bool&optVar)
  {
          if(str=="on")   {optVar = true; return true;}
          if(str=="off")  {optVar = false;return true;}
          if(str=="1")    {optVar = true; return true;}
```

```
            if(str=="0")   {optVar = false;return true;}
            return false;
    }
    bool Read(string& str,string&optVar)
    {
            optVar = str;
            return true;
    }


};


#ifdef __C_OPTIONS_TEST
void main(int  pn,char*ps[])
{
  COptions Options(pn,ps);

  float  ftest;
  int    itest;
  string inputFile;
  string outputFile;
  bool   btest;

  try       {
    Options.Parse("-fov"          ,ftest          ,"Fild Of View"       ,3.1f);
    Options.Parse("-in"           ,inputFile      ,"Input File Name");
    Options.Parse("-out"          ,outputFile     ,"Output File Name"   ,string("out.file"));
    Options.Parse("-numViews"  ,itest          ,"Number Of Views"  ,984);
    Options.Parse("-rf"           ,btest          ,"RingFix (on,off)"   ,true);
    }
  catch(...)
  {
    Options.Usage("Catch error");exit(0);
  }
  if(!Options.IsOk()){Options.Usage("Parse Error");exit(0);}

}
#endif //C_OPTIONS_TEST
```