Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования "Пермский национальный исследовательский политехнический университет"

Электротехнический факультет

Информационные технологии и автоматизированные системы

Курсовая работа

по дисциплине: «Алгоритмические языки программирования»

**АНАЛИЗ ВИДЕО. ВОССТАНОВИТЬ ТРАЕКТОРИЮ ОБЪЕКТА В 3D – ПРОСТРАНСТВЕ ПО ВИДЕОПОТОКУ С КАМЕРЫ**

|  |
| --- |
| Выполнили:  студенты гр. АСУ2-19-1м  Князев Александр Игоревич Боброва Ирина Александровна |
| Проверил: доцент кафедры ИТАС  Курушин Даниил Сергеевич |

Пермь 2019

Содержание

[Введение 3](#_Toc22509761)

[**I.** **Анализ методов распознавания лица на изображении** 4](#_Toc22509762)

[**О нейронных сетях** 4](#_Toc22509763)

[**Использование нейронных сетей в OpenCV** 5](#_Toc22509764)

[**Поиск быстрых методов обнаружения человека на изображении** 5](#_Toc22509765)

[**О каскадах HAAR** 6](#_Toc22509766)

[**Использование каскадов HAAR в OpenCV** 6](#_Toc22509767)

[**О HOG-дескрипторе** 7](#_Toc22509768)

[**Использование HOG-дескриптора в OpenCV** 8](#_Toc22509769)

[**II.** **Оптимизация программы** 9](#_Toc22509770)

[**Трекинг** 9](#_Toc22509771)

[**Использование Mosse трекинга в OpenCV** 12](#_Toc22509772)

[**III.** **Выбор средств реализации** 13](#_Toc22509773)

[**Обзор** 13](#_Toc22509774)

[**Описание системы** 13](#_Toc22509775)

[**Программная реализация** 17](#_Toc22509776)

[**Тестирование** 22](#_Toc22509777)

[Заключение 23](#_Toc22509778)

[**Список литературы** 25](#_Toc22509779)

[**Приложение** 27](#_Toc22509780)

[show\_paths.py 30](#_Toc22509781)

# Введение

Компьютерное зрение связано с автоматическим извлечением и анализом полезной информации из одного изображения, либо последовательности изображений. Оно включает разработку теоретической и алгоритмической основы для достижения автоматического визуального понимания. Изучение этих проблем является как теоретическим, так и практически важным.

В рамках данной курсовой работы была поставлена задача восстановления траектории объекта по видеопотоку с камеры в 3D-пространстве. Задача восстановления траектории с видеопотока сложна по ряду причин.

1. Главная сложность заключается в необходимость связать плоскости наблюдения камеры и плана. Необходим расчет координат объекта и его перемещения в одной плоскости и перенесение их на план, представляющий собой вид помещения в другой плоскости.

2. Искажения. Как и любая оптическая система, камера вносит искажения, которые приводят к погрешностям при построении траектории на плане.

3. Привязка поля зрения камеры к видимой области. Следует создать алгоритм, который сможет самостоятельно вычислить область по опорным предметам, попадающим в объектив камеры. Привязать можно и вручную, однако это не является продуктивным решением в больших системах.

На начальном этапе были обозначены следующие задачи:

1. выбрать наиболее оптимальный способ распознавания объекта;
2. определить инструмент отслеживания (трекинговая система);
3. решить, как именно будет рассчитываться расстояние до объекта;
4. провести испытания системы.
5. **Анализ методов распознавания лица на изображении**

Изначально планировалось осуществлять нахождение людей на изображении при помощи нейронных сетей. Нейронные сети довольно хорошо зарекомендовали себя в последнее время в области компьютерного зрения, поскольку правильно натренированные нейронные сети отлично справляются с задачами распознавания человека на изображении.

**О нейронных сетях**

Нейронная сеть – это последовательность нейронов, соединенных между собой синапсами. Структура сети и нейрона пришла в программирование из биологии. Нейронные сети используются для решения сложных задач, которые требуют аналитических вычислений подобных тем, что делает человеческий мозг. Нейрон — это вычислительная единица, которая получает информацию, производит над ней простые вычисления и передает ее дальше. В том случае, когда нейронная сеть состоит из большого количества нейронов, вводят термин слоя.

Работа нейронной сети основана на весах синапсов. Синапс - это связь между двумя нейронами. Информация передаваемая от одного нейрона к другому умножается на вес синапса, после чего складывается с произведениями других информаций и весов синапсов входящих в рассматриваемый нейрон, затем полученная сумма используется в качестве аргумента некоторой функции, вычисление которой и даст результат работы нейрона. Полученный результат передается последующим нейронам, находящимся в следующем слое. Первый слой нейронов является входным. Последний слой преобразует полученные данные в необходимый нам вид. Обучение сети основано на изменении весов синапсов в соответствии с ошибкой результата от предполагаемого. При работе с распознаванием образов используются слои convolution, pooling, а также стандартные Fully Connected слои.

**Использование нейронных сетей в OpenCV**

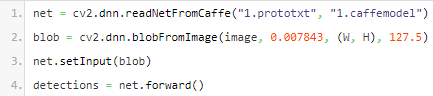


Рис 1. Код использования нейронных сетей на языке программирования Python

Строка 1 отвечает за загрузку нейронной сети, отправку информации в нейронную сеть и получение информации из нее. Файл 1.prototxt содержит в себе структуру нейронной сети, а файл 1.caffemodel содержит в себе информацию о весах нейронов сети, этот файл был получен путем тренировки сети на основе файла 1.prototxt, а также наборе изображений, соответствующих каждому из описанных ранее классов. Оба эти файла были сформированы заранее.

В строке 2 формируются настройки для нейронной сети, которые затем используются при запуске нейронной сети в строке 3.

Запись полученных координат прямоугольников, содержащих изображения людей, происходит в строке 4.

**Поиск быстрых методов обнаружения человека на изображении**

Однако минус нейронных сетей в том, что при обработка изображения происходит довольно долго на устройствах не имеющих больших мощностей. Поиск людей на картинке с использованием Raspberry Pi занимал несколько секунд для каждого кадра.

Было принято решение найти другой более быстрый метод поиска людей на изображении. В ходе анализа разных методов было выделено два наиболее подходящих: каскады HAAR и HOG-дескрипторы.

Быстрее работали каскады HAAR, однако они порой уступали в точности HOG-дескрипторам, которые работали немного дольше. В программе представлены все три метода поиска человека на изображении, для того, чтобы пользователь имел возможность выбирать нужный ему метод исходя из требуемой точности и скорости.

**О каскадах HAAR**

Основное преимущество детектора Хаара: скорость. Благодаря быстрой обработке изображения, можно с лёгкостью обрабатывать потоковое видео. Детектор Хаара используется для распознавания большинства классов объектов. К ним относятся лица и другие части тела людей, номера автомобилей, пешеходы, дорожные знаки, животные и.т.д

Классификатор формируется на примитивах Хаара путём расчёта значений признаков. Для обучения на вход классификатора сначала подаётся набор «правильных» изображений с предварительно выделенной областью на изображении, дальше происходит перебор примитивов и расчёт значения признака. Вычисленные значения сохраняются в файле в формате xml.

**Использование каскадов HAAR в OpenCV**

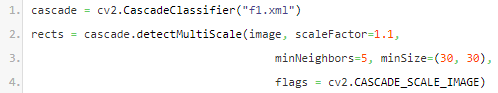


Рис 2. Код использования каскадов HAAR на языке программирования Python

В строке 1 происходит загрузка каскадов в функцию каскадов HAAR. Файл с каскадами был создан ранее путем обработки большого количества изображений, на которых присутствовали люди и отсутствовали. Процесс создания такого файла не представляет трудности, однако занимает большое количество времени и требует правильную подборку изображений, для того, чтобы поиск людей на изображении осуществлялся правильно.

Строки 2-4 отвечают за получение координат прямоугольников, содержащих в себе фрагметы изображения с людьми.

**О HOG-дескрипторе**

Для обнаружения людей на кадре используется алгоритм вычисления HOG (гистограмма направленных градиентов) признаков.

Цвет изображения не имеет значения, поэтому изображение приводится к черно-белому виду.

Изображение разбивается на блоки (16х16 пикселей). Для каждого блока вычисляются направления градиентов. Под градиентом здесь понимается аппроксимация градиента функции интенсивности (яркости), которая предполагается дифференцируемой, но известной лишь в узлах равномерной сетки – пикселях, в заданной точке с помощью некоторой разностной схемы.

После этого рассматриваемый блок заменяется стрелкой с направлением, преобладающем в этом квадрате.

Чтобы обнаружить людей на преобразованном изображении, требуется найти такие участки, которые наиболее похожи на известную HOG-структуру, полученную в результате обучения. В своей программе мы используем заранее натренированную структуру, содержащуюся в библиотеке OpenCV.

**Использование HOG-дескриптора в OpenCV**

Для того, чтобы использовать HOG-дескрипторы, необходимо импортировать библиотеку cv2. Алгоритм вычисления HOG-признаков реализован в классе HOGDescriptor, входящем в состав модуля objdetect.

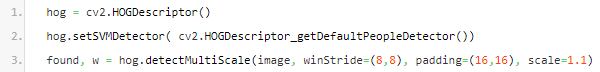


Рис 3. Код использования HOG-дескриптора на языке программирования Python

В строке 1 происходит инициализация HOG-дескриптора в переменную hog.

Строка 2 отвечает за выбор детектора, настроенного на поиск конкретных объектов.

В строке 3 функция detectMultiScale принимает на вход этой функции подается изображение, шаг окна, padding и коэффициент увеличения поискового окна. Переменная found в результате работы функции detectMultiScale содержит в себе массив из координат x и y, а также ширины и высоты найденных объектов. На основе этих данных на выходном изображении рисуется рамочка, выделяющая найденный объект. Переменная w содержит веса найденных объектов.

1. **Оптимизация программы**

Несмотря на увеличившуюся скорость поиска, обработка каждого изображения всеравно занимала порядочное количество времени. Нужно было придумать такой подход, который бы сделал обработку еще быстрее.

В качестве решения проблемы обработки каждого кадра, было предложено обрабатывать каждый 20-й кадр, а в промежутке между кадрами отслеживать положение обнаруженных на изображении людей. Как показали эксперименты, функции, отслеживающие объект на изображении, работают значительно быстрее функций поиска объекта на изображении.

**Трекинг**

Трекинг объектов – отслеживание перемещения объектов в пространстве. Существуют различные методы реализации трекинга объектов на видео. При трекинге объекта важна скорость работы алгоритма и его точность. Точность выражается в том, чтобы верно определять новое местоположение объекта, а также верно определять новые объекты и прекратить отслеживание объектов, покинувших поле видимости камеры наблюдения.

В нашем проекте трекинг осуществляется в связке с алгоритмами обнаружения людей на изображении. Координаты человека и границы его силуэта определяются алгоритмами поиска людей, после чего координаты передаются алгоритму, отвечающему за трекинг.

Далее будут кратко рассмотрены восемь алгоритмов отслеживания объектов, встроенных в OpenCV, а также продемонстрированы варианты использования каждого из этих трекеров. Наконец, будут рассмотрены результаты каждого из трекеров, отметив, какие из них работали в каких ситуациях, а какие - нет.

Краткое описание каждого алгоритма работы отслеживания объектов приведены ниже:

BOOSTING Tracker: Основан на онлайн-версии AdaBoost - алгоритма, который использует детектирование лиц на основе каскадов Хаара. Трекер достаточно медленный и производительность отслеживания посредственная, достоверно не знает, когда отслеживание удалось. Интересен только по унаследованным причинам и сравнением с другими алгоритмами. (версия OpenCV не позднее 3.0.0)

MIL Tracker: Имеются сходства с трекером BOOSTING. Большая разница в том, что вместо того, чтобы рассматривать только текущее местоположение объекта в качестве положительного примера, он просматривает небольшую окрестность вокруг текущего местоположения, чтобы сгенерировать несколько потенциальных положительных примеров. Лучшая точность, чем BOOSTING tracker, но плохо сообщает о сбоошибка отслеживания не сообщается достоверное. (минимум OpenCV 3.0.0)

KCF Tracker: Множество положительных образцов, используемых в трекере MIL, имеют большие перекрывающиеся области. Эти перекрывающиеся данные приводят к некоторым хорошим математическим свойствам, которые используются трекеров KFC, чтобы сделать отслеживание одновременно более быстрым и точным. Точность и скорость выше, чем у MIL, и он сообщает о сбое отслеживания лучше, чем BOOSTING и MIL. Из минусов можно выделить отсутствие восстановления после полной окклюзии (отслеживаемый объект полностью сокрыт другим объектом). (Доступен версиях не раньше OpenCV 3.1.0)

CSRT Tracker: Обеспечивает увеличение, локализацию выбранной области и улучшенное отслеживание непрямоугольных областей или объектов. Трекер использует только 2 стандартных функции (HoGs и Colornames). Он также работает со сравнительно более низкой частотой кадров (25 кадров в секунду), но обеспечивает более высокую точность отслеживания объектов. Точнее, чем KCF, но немного медленнее. (Версия минимум OpenCV 3.4.2)

MedianFlow Tracker: Внутренне этот трекер отслеживает объект во времени как в прямом, так и в обратном направлении и измеряет расхождения между этими двумя траекториями. Хорошо работает, когда движение предсказуемо и мало, сообщая об ошибках; однако, если имеется слишком большой скачок в движении, например, быстро движущиеся объекты или объекты, которые быстро меняются по своему внешнему виду или же присутствуют окклюзии, модель потерпит неудачу. (минимум OpenCV 3.0.0)

TLD Tracker: Данный трекер разбивает долгосрочную задачу отслеживания на три компонента - (краткосрочное) отслеживание, обучение и обнаружение. Трекер следит за объектом от кадра к кадру. Детектор локализует все наблюдения, которые наблюдались до сих пор, и корректирует трекер, если это необходимо. Обучение оценивает ошибки детектора и обновляет его, чтобы избежать этих ошибок в будущем. Трекер сильно подвержен колебаниям, например, может отслеживать соседний объект, а не выбранный. С положительной стороны, этот трекер отлично справляется с окклюзиями. Если у вас есть видеопоследовательность, где объект скрыт за другим объектом, этот трекер может быть хорошим выбором. (минимум OpenCV 3.0.0)

GOTURN Tracker: Из всех алгоритмов слежения в классе трекеров это единственный, основанный на сверточной нейронной сети (CNN). Данная нейронная сеть надежна для просмотра изменений, изменений освещения и деформаций, но это не очень хорошо справляется с окклюзией [8]. Для запуска требуются дополнительные файлы моделей. (минимум OpenCV 3.2.0)

MOSSE Tracker: Трекер устойчив к изменениям освещения, масштаба, позы и нежестких деформаций. Он также обнаруживает окклюзию на основе соотношения пиковых и боковых лепестков, что позволяет трекеру приостанавливать и возобновлять работу с того места, где он остановился, когда объект снова появляется. MOSSE tracker также работает с более высокой частотой кадров (450 кадров в секунду и более). Его легко реализовать, так же точно, как другие сложные трекеры, но работать он будет гораздо быстрее. В плане производительности отстает от трекеров, основанных на глубоком обучении. Не такой точный, как CSRT или KCF, но хороший выбор, если вам нужно быстродействие. (минимум OpenCV 3.4.1).

После сравнительного анализа было решено остановиться на функции трекинга MOSSE, поскольку она работала быстрее остальных и не сильно уступала в точности. Однако в случаях, когда требуется высокая точность используется функция трекинга KCF, так как на данный момент это самая точная функция трекинга, учитывающая изменение размеров объекта, его перекрытие другими объектами и различные искажения.

**Использование Mosse трекинга в OpenCV**

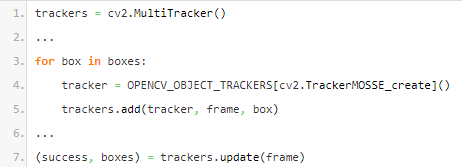


Рис 4. Код использования трекинга Mosse на языке программирования Python

В строке 1 происходит инициализация функции множественного трекинга в переменную trakers. Данная процедура выполняется каждый раз после обнаружения людей на изображении. К сожалению, в библиотеке OpenCV не предусмотрено удаление конкретных трекеров, поэтому приходится чистить весь массив трекеров, заново инициализируя его.

После обнаружения объектов и обработке результатов, выданных функциями детектирования, информация об обнаруженных объектах записывается в массив boxes. Для каждого объекта инициализируется свой трекер. Созданный трекер добавляется в общий массив трекеров с переменной, содержащей, изображение, а также координаты, ширину и высоту отслеживаемого объекта. Строки кода 3 – 5 выполняются только после того, как было заново проведено обнаружение людей на кадре.

В строке 7 происходит обновление позиций отслеживаемых объектов. После этого на новом кадре объекты обводятся рамочками.

1. **Выбор средств реализации**

**Обзор**

При решении задачи анализа методов обнаружения лиц, были рассмотрены: искусственные нейронные сети, алгоритмы поиска на основе каскадрв Хаара и метод Hog-дескрипторов [1][2]. После тестирования были получены средние значения времени, затраченного на обработку одного кадра. Больше всего времени на поиск лиц, было затрачено нейронной сетью. Для обработки изображений на микроконтроллерах этот показатель критичен. Скорость каскадов Хаара и HOG-детектора выше, но точность ниже чем у нейронных сетей. Компенсировать этот недостаток можно при помощи создания более качественного и оптимизированного датасета для каскадов Хаара. Реализаций метода HOG для микроконтроллеров с возможностью обучения найти не удалось.

## **Описание системы**

В связи с этим было принято решение остановиться на каскадах Хаара, а для снижения среднего времени обработки изображения, применить алгоритм трекинга обнаруженных силуэтов. Такие алгоритмы значительно меньше нагружают микроконтроллер и достаточно быстры.

Библиотека компьютерного зрения OpenCV поддерживает около десятка различных функций трекинга [3]. После сравнительного анализа было решено остановиться на функции трекинга MOSSE, поскольку она работала быстрее остальных и не сильно уступала в точности. Однако в случаях, когда требуется высокая точность используется функция трекинга KCF, так как на данный момент — это самая точная функция трекинга, учитывающая изменение размеров объекта, его перекрытие другими объектами и различные искажения.

Большинство изученных нами методов при измерении расстояния до объекта отталкиваются от того, то камера расположена параллельно земле. Однако такой подход не применим для камер видеонаблюдения, расположенных в помещении. Поэтому необходимо использовать подход, учитывающий угол наклона камеры.

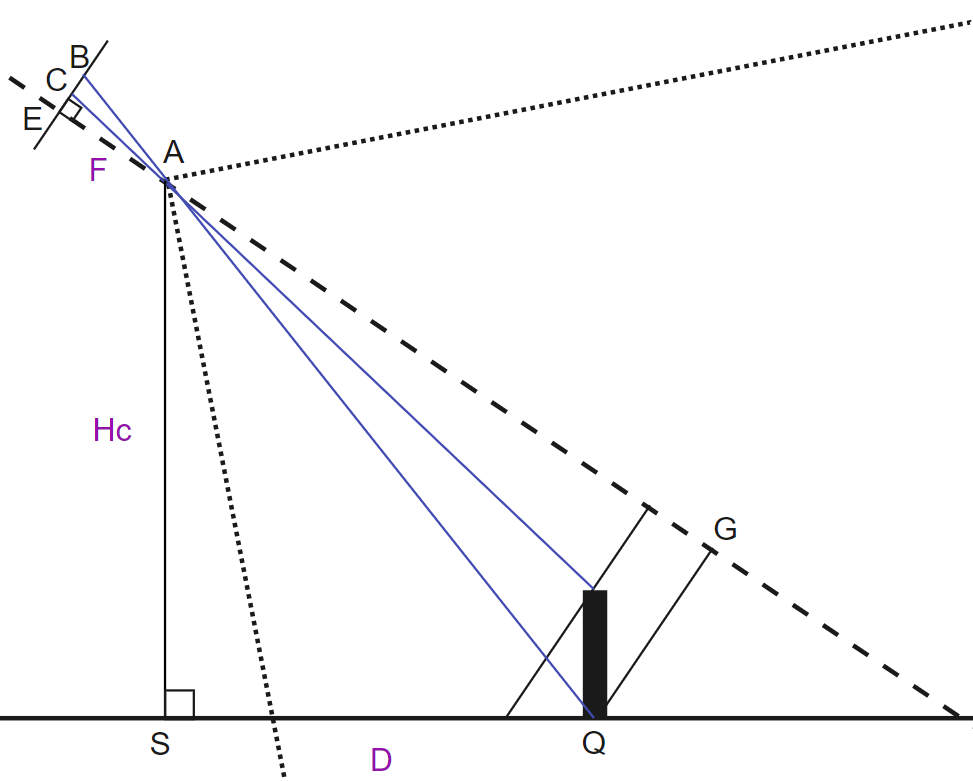


Рис. 5. Расположение камеры и человека

Для решения задачи поиска расстояния от камеры до объекта, решили использовать следующую формулу:

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

Где:

D – расстояние от основания крепления камеры до объекта;

– угол наклона камеры;

h – высота изображения, полученного с камеры;

– нижняя координата человека на изображении (в пикселях);

F – фокусное расстояние;

Hc – высота, на которой расположена камера.

Формула была получена следующим образом:

Рассмотри треугольник EAB, он прямоугольный. Катеты EB и EA известны, поэтому угол EAB можно найти по формуле (2).

|  |  |  |
| --- | --- | --- |
|  |  | (2) |

Затем рассмотрим треугольник SAQ, он также прямоугольный. Сторона SQ = D, а SA = Hc. Чтобы найти D, необходимо найти угол SAQ и воспользоваться формулой тангенса (3).

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

Чтобы найти угол SAQ, необходимо из угла SQG, который нам известен, и является углом отклонения камеры от вертикальной оси, вычесть угол QAG, который равен углу EAB (4).

|  |  |  |
| --- | --- | --- |
|  |  | (4) |

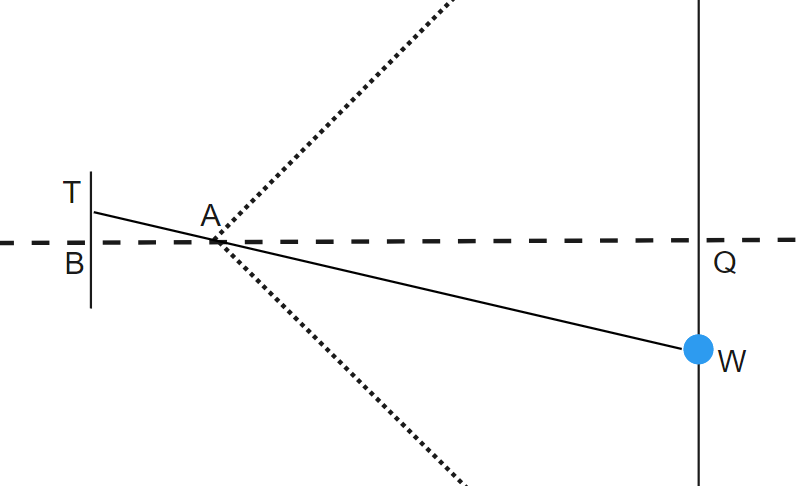


Рис. 6. Расположение камеры и человека вид сверху

Смещение по оси Х вычисляется по следующей формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (5) |

Где:

X – расстояние от левого края изображения до объекта;

D – расстояние от основания крепления камеры до объекта;

h – высота изображения, полученного с камеры;

w – ширина изображения, полученного с камеры;

– нижняя координата человека на изображении (в пикселях);

F – фокусное расстояние;

Hc – высота, на которой расположена камера;

x – координата человека на изображении (в пикселях).

Формула была получена следующим образом:

Треугольники TBA и WQA подобные, следовательно, отклонение X, равное WQ можно найти по формуле (6).

|  |  |  |
| --- | --- | --- |
|  |  | (6) |

Рассмотрим треугольник AQS рисунок 5. Найдем AQ по формуле (7).

|  |  |  |
| --- | --- | --- |
|  |  | (7) |

Рассмотрим треугольник ABE рисунок 5. Найдем AB по формуле (8).

|  |  |  |
| --- | --- | --- |
|  |  | (8) |

При помощи выведенных формул получаем формулу (5).

## **Программная реализация**

В системе реализовано несколько подходов к распознаванию, а именно: каскады Хаара, HOG-дескрипторы и нейронная сеть.

Полный код программы можно посмотреть в личном репозитории группы на github.

Вначале импортируем модули (стандартные и созданные нами), производим базовые настройки параметров окна и времени ожидания.

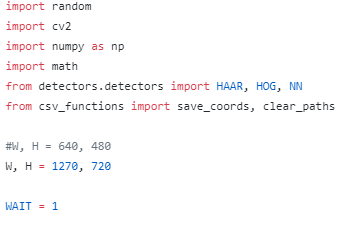


Рис. 7. Базовые настройки

Преобразуем цветовой режим изображения в оттенки серого, создаем каскад и определяем лица на изображении, определяем размеры детектирующего окна.

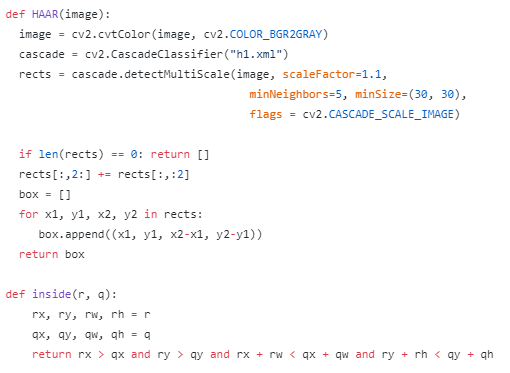


Рис. 7. Начало реализации методом каскадов Хаара

Так же преобразуем цветовой режим изображения. Создаем экземпляр объекта cv2.HOGDescriptor() как hog. После создания объект имеет все связанные атрибуты и методы класса cv2.HOGDescriptor(). Устанавливаем параметры для детектирующего прямоугольника.

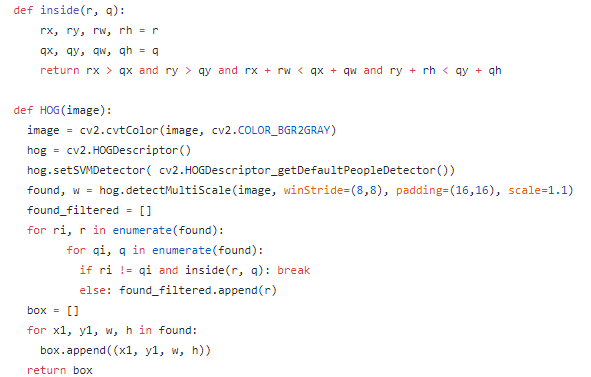


Рис. 8. Начало реализации методом HOG-дескрипторов

Инициализируем метки CLASSES, загружаем модель. Затем следует преобразование кадра в blob с модулем dnn. Устанавливаем blob как входные данные в нашу нейросеть и передаём эти данные через net, которая обнаруживает объекты. Прописываем условия, при которых объект будет точно детектироваться.

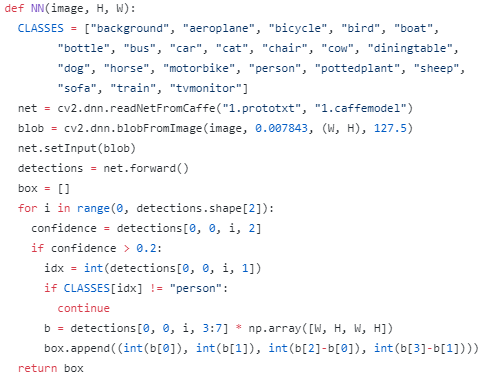


Рис. 9. Начало реализации детектирования посредством нейросети

Следующая функция нужна для того, чтобы соотнести новые координаты объектов со старыми. Поскольку при вызове детектирующей функции, порядок объектов в массиве может измениться. Рассчитывается расстояние между текущим положением и старыми. Таким образом находится индекс элемента в старом массиве. Если же расстояние превышает 150 пикселей, то объект скорее всего является новым и обрабатывается как новый.

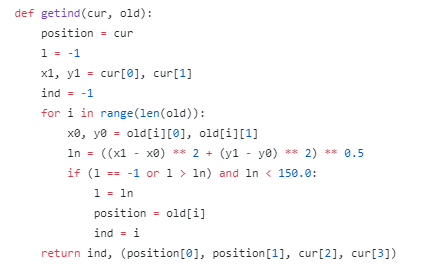


Рис. 10. Определение позиции объекта и сохранение координат

Формулы, описанные в разделе «Описание системы», представлены в следующем фрагменте кода.

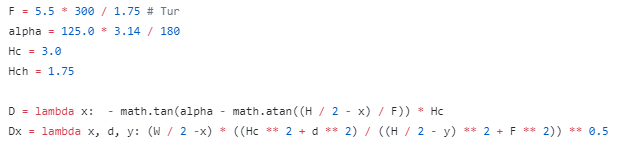


Рис. 11. Нахождение значений

Если был получен фрейм, изменяем размеры изображения и обрабатываем (ищем людей) каждый 20 кадр. Затем с помощью описанной выше функции getind() сопоставляем координаты объектов и обрабатываем новые. Чтобы объект не терялся, в случае, если детектор не распознает его, у всех распознанных объектов есть «время жизни» - количество распознаваний, которое дается, на повторный поиск объекта.

В остальное время используем алгоритмы трекинга. Создаем новый объектный трекер для ограничительной рамки и добавляем его в наш мульти-объектный трекер. Иначе получаем обновленные координаты ограничивающего прямоугольника для каждого отслеживаемого объекта.

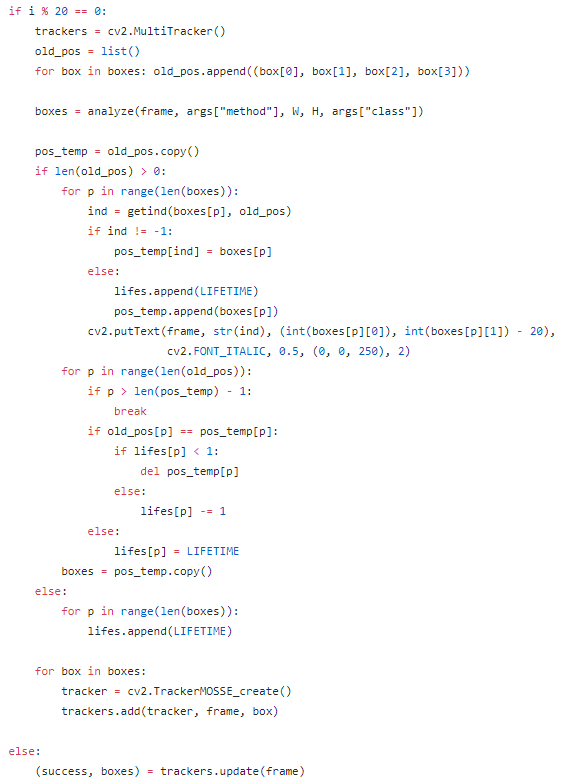


Рис. 12. Обработка изображений нейросетью

Далее записываются и отображаются координаты отслеживаемых прямоугольников. Программа прерывается при нажатии клавиш “q”, либо “esc”.

Вызывается модуль, отвечающий за отображение траектории. Для отображения траекторий, используется библиотека matplotlib. Берутся все файлы из папки paths и считываются координаты для построения траекторий движения. Для плавности построения используется метод среднего скользящего.

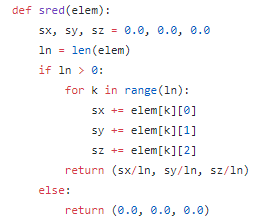


Рис. 13. Метод скользящего среднего

Решение использовать вместе каскады Хаара, HOG-дескрипторы и нейросеть было принято в ходе практической реализации и анализа временных затрат. Скорость каскадов Хаара и HOG-детектора выше, чем у нейронной сети, но точность детектирования ниже. Оптимальным решением было добавить все способы и использовать их в зависимости от требований ситуации. Когда точность является критичным показателем используем нейросеть, когда важна скорость отработки алгоритма – используем метод каскадов Хаара и HOG-дескрипторы.

## **Тестирование**

Разработанный прототип программы была протестирована на тестовой подборке видео. Результаты измерения координат имели максимальную погрешность 0,35 м.

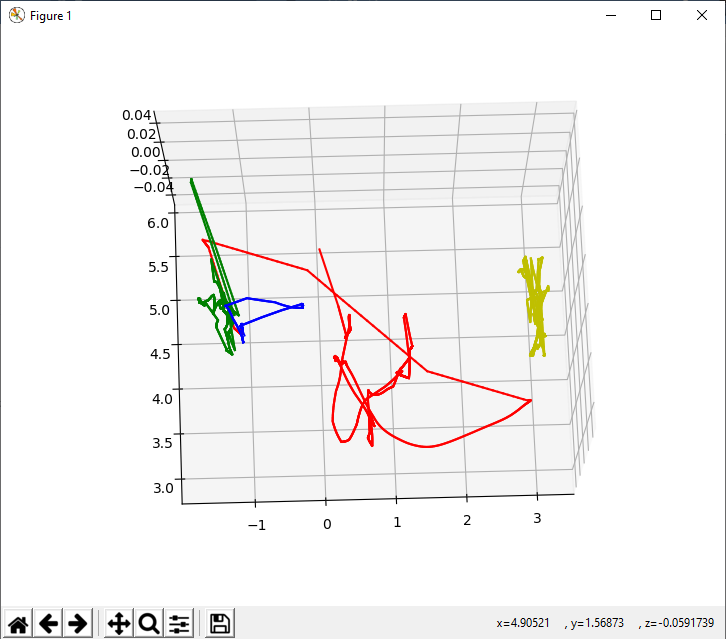


Рис. 14. Проверка работоспособности системы (отображение пути)

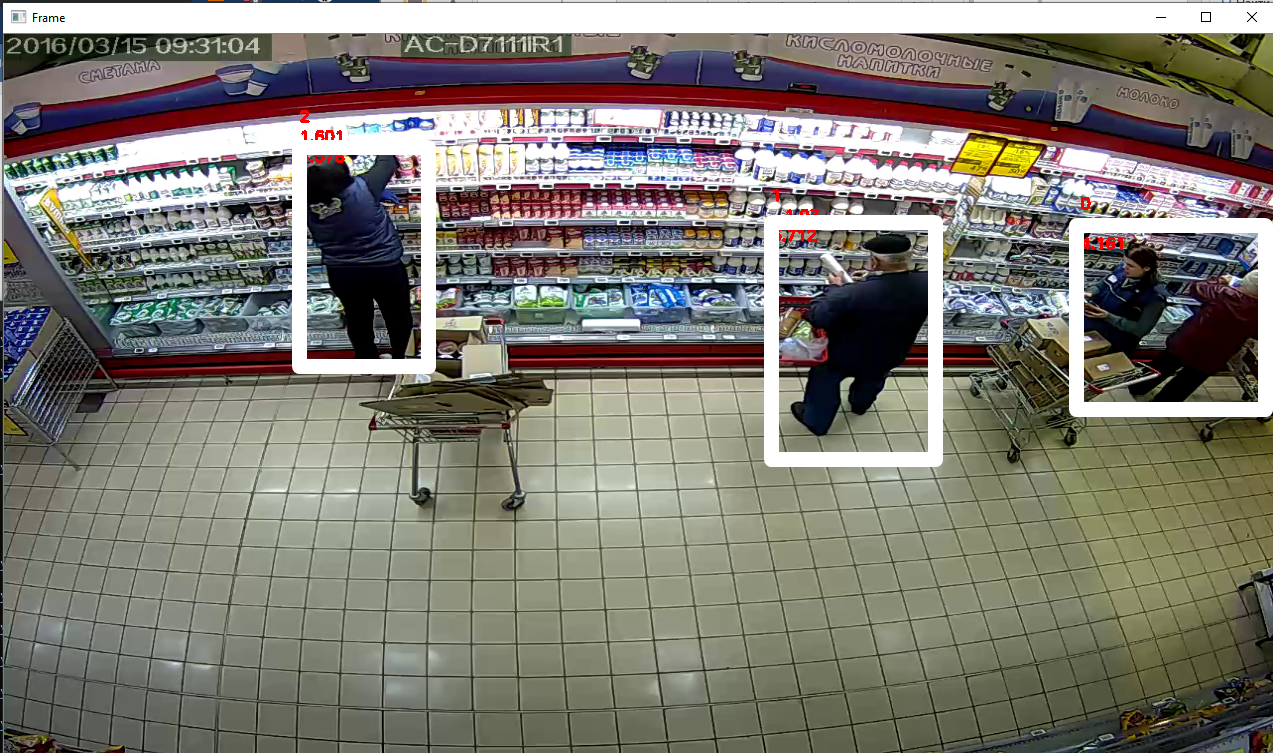


Рис. 15. Проверка работоспособности системы (поиск объектов)

# Заключение

В результате научного исследования были выполнены все обозначенные задачи. Главная сложность построения траектории объекта заключалась в необходимость связать плоскости наблюдения камеры и плана. Расчет координат объекта производился математическим методом и описан в разделе “Описание системы”. Касательно искажений было принято решение о допущении небольших погрешностей, если для нас не критичен данный показатель. В том случае если этот показатель важен необходимо провести преобразование кадров, но данный способ решения проблемы требует дополнительных временных затрат. Область зрения была определена исходя из среднего угла обзора в 90°, для вычисления положения объекта рост человека был взят как средний.

В ходе практической реализации было принято решение использовать каскады Хаара, HOG-дескрипторы и нейросеть. В зависимости от ситуации выбирать наиболее оптимальный способ. В качестве системы отслеживания был выбрать трекинг Mosse как наиболее оптимальный и хорошо показавший себя в ходе тестовых испытаний.

Был разработан базовый функционал системы отслеживания, а также определены перспективы для дальнейшей работы.

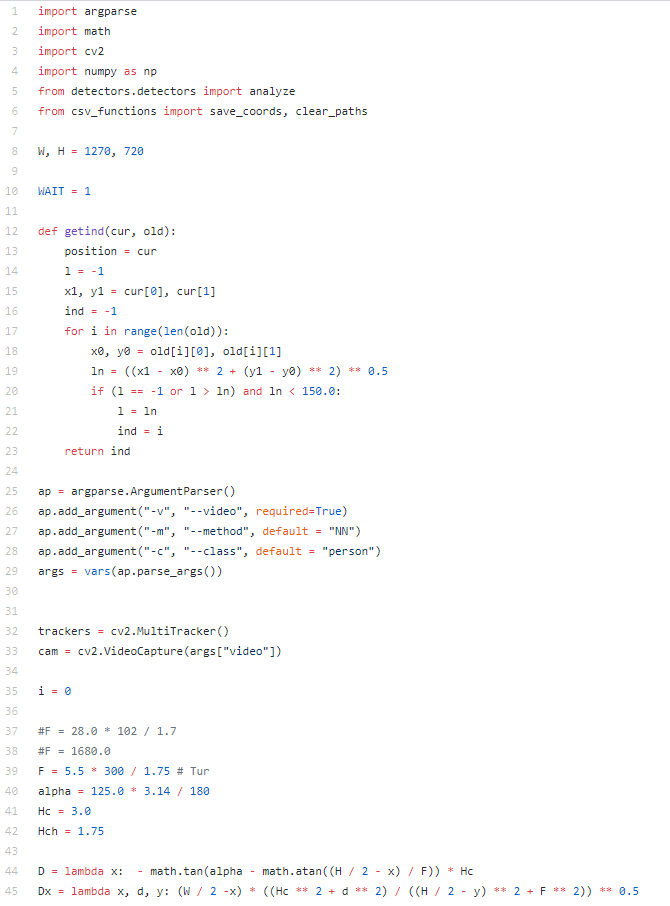
Продолжением настоящего исследования будет являться сборка системы со всеми базовыми функциями, оптимизация алгоритмов, финальное тестирование и исправление ошибок.

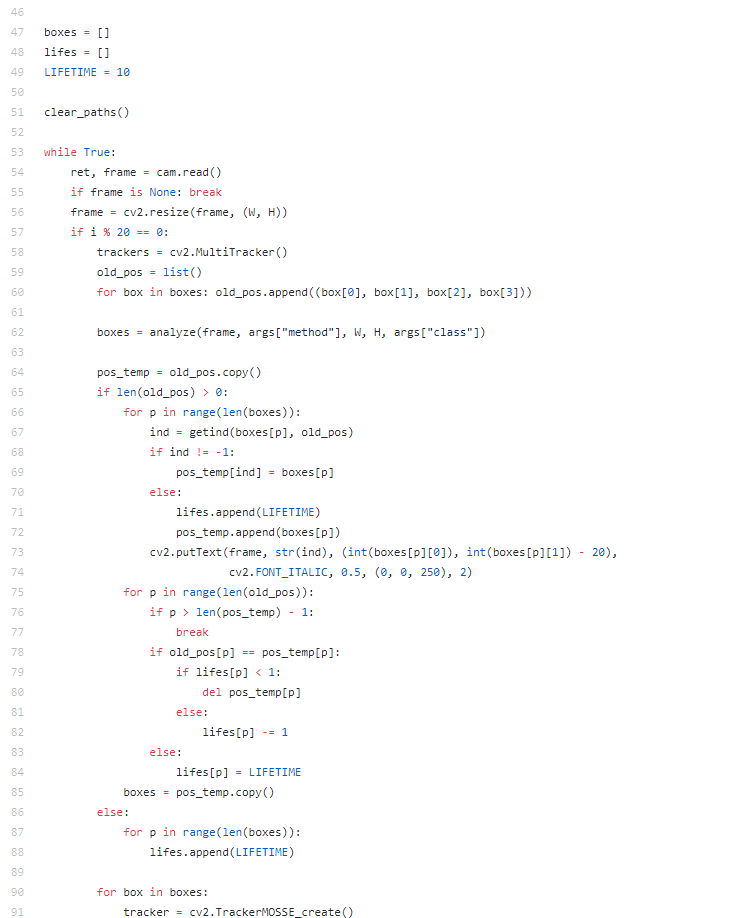
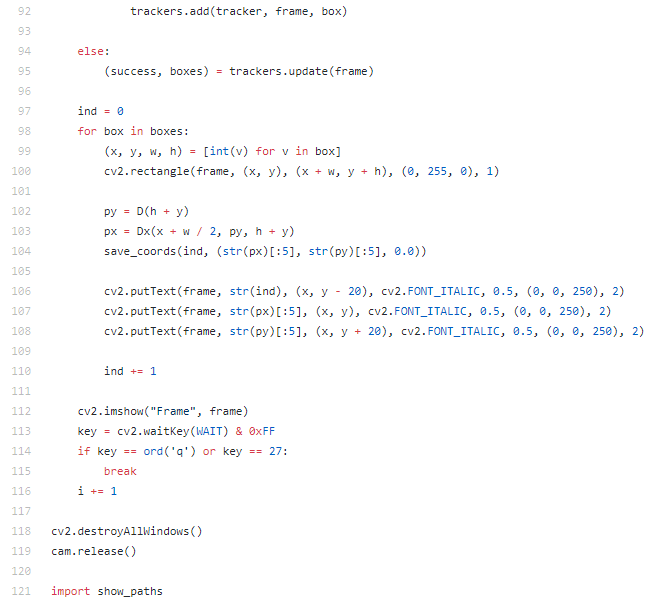
## **Список литературы**

1. Kokoulin A.N., Tur, A.I., Yuzhakov A.A., Knyazev A.I. Hierarchical convolutional neural network architecture in distributed facial recognition system  // Proceedings of the 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering, ElConRus 2019
2. Тур А.И., Кокоулин А.Н. Применение каскадов Хаара для распознавания объектов // АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ УПРАВЛЕНИЯ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ Материалы всероссийской научно-технической конференции. В 2-х томах. 2018. С 98-103.
3. Обучение OpenCV каскада Хаара. URL: <https://m.habr.com/ru/post/208092/> (дата обращения: 07.10.2019).
4. SATYA MALLICK. Object Tracking using OpenCV (C++/Python) URL: https://www.learnopencv.com/object-tracking-using-opencv-cpp-python (07.10.2019)
5. Adrian Rosebrock. Tracking multiple objects with OpenCV URL: https://www.pyimagesearch.com/2018/08/06/tracking-multiple-objects-with-opencv/ (07.10.2019)
6. Navneet Dalal and Bill Triggs. Histograms of Oriented Gradients for Human Detection URL: http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf (07.10.2019)
7. Histogram of oriented gradients URL: https://en.wikipedia.org/wiki/Histogram\_of\_oriented\_gradients (08.10.2019)
8. [Satya Mallick](https://www.learnopencv.com/author/spmallick/). GOTURN : Deep Learning based Object Tracking URL: <https://www.learnopencv.com/goturn-deep-learning-based-object-tracking> (дата обращения: 13.10.2019)
9. Adrian Rosebrock. Building an Image Hashing Search Engine with VP-Trees and OpenCV URL: https://www.pyimagesearch.com/2019/08/26/building-an-image-hashing-search-engine-with-vp-trees-and-opencv (дата обращения: 13.10.2019)
10. Александр Коробков. Отслеживание объектов в видеопотоке. Методы построения траекторий URL: <http://secuteck.ru/articles2/videonabl/otslezhivanie-obektov-v-videopotoke-metody-postroeniya-traektoriy> (дата обращения: 13.10.2019)

## **Приложение**

main.py



show\_paths.py

