**Санкт-Петербургский политехнический университет Петра Великого**
**ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И ТЕХНОЛОГИЙ**
**ВЫСШАЯ ШКОЛА КИБЕРФИЗИЧЕСКИХ СИСТЕМ И УПРАВЛЕНИЙ**

Работа допущена к защите
Руководитель ООП
А.А. Ефремов
«___» _____ 2017 г.

ВЫПУСКНАЯ РАБОТА БАКАЛАВРА

# Система анализа и распознавания изображений

Направление: 09.03.02 – Информационные системы и технологии

Выполнил:

студент гр. 43502/2 _____ Салех Л.О.А.

подпись, дата

Руководитель:

доцент, к. т. н. _____ Хлопин С.В.

подпись, дата

Санкт-Петербург
2017

# СИСТЕМА АНАЛИЗА И РАСПОЗНАВАНИЯ ИЗОБРАЖЕНИЙ

***Аннотация:*** Цель данной работы - изучить алгоритм распознавания предметов, изучить библиотеку алгоритмов компьютерного зрения и обработки изображений с открытым исходным кодом Open CV, отслеживание и распознавание объектов с использованием библиотеки Open CV, проведение анализа и сделать выводы о результатах анализа. Этапы выполнения работы включают изучение предметной области, выбор инструментария, исследование алгоритма, изучение основных методов и функций библиотеки OpenCV, реализация кодов программы и анализ полученных результатов.
***Ключевые слова:*** компьютерное зрение, детектирование объектов, библиотека opencv, отслеживание объектов, анализ изображения

# SYSTEM OF ANALYSIS AND RECOGNITION OF IMAGES

***Abstract:*** The purpose of this work is to study the algorithm of object recognition, to study the library of computer vision and image processing algorithms with open source Open CV, to track and recognize objects using the Open CV library, to carry out analysis and draw conclusions about the results of the analysis. The stages of the work include the study of the subject area, the choice of tools, the study of the algorithm, the study of the main methods and functions of the Open CV library, the implementation of program codes and the analysis of the results.
***Keywords:*** computer vision, object detection, opencv library, object tracking, image analysis

# Оглавление

# Введение

Огромное внимание в настоящее время уделяется системам, использующим машинное зрение в качестве основного источника информации. Для этого ведутся поиски новых алгоритмов обработки и распознания изображений.

Одной из областей информатики, достижения которой можно использовать уже сейчас является компьютерное зрение. Алгоритмы для нахождения границ единичных объектов, детектирование групп различных объектов, будь то анализ человеческих лиц или автоматическое определение номеров автомобилей, являются лишь малой частью достижений данной науки.

Компьютерное зрение как научная дисциплина появилась в конце 1970-х годов. Основная цель компьютерного зрения — автоматизация процесса распознавания и обработки цифровых изображений и видеофайлов. Данная дисциплина нашла применение во множестве областей, касающихся обработки изображений и распознавания образов в промышленных, военных и бытовых системах. Например, одним из больших достижений в этой области является автоматизация процесса вождения автомобиля.

Основным применением компьютерного зрения является распознавание образов, которое в науке принято называть сегментацией. Сегментация - это группирование пикселей изображений по определенным признакам. После сегментации изображения проводится детектирование объектов, присутствующих в данном изображении. Существует множество методов детектирования элементов изображений, которые имеют свою классификацию:

1. Детекторы, использующие градиент яркости. (фильтр Собеля)
2. Детекторы, использующие коэффициент совпадения с шаблоном, которые применяются для нахождения объектов по определенным образцам, взятым из других изображений.

3. Детекторы, использующие преобразование пространства, используются для нахождения предметов, когда они искривлены геометрически.
4. Детекторы, использующие статистические методы.

Задача распознавания объектов и образов представляет собой одно из приоритетных направлений развития алгоритмов машинного обучения и компьютерного зрения.

# Постановка задачи

**Цели работы:**

- Изучить алгоритм распознавания предметов.
- Изучить библиотеку алгоритмов компьютерного зрения и обработки изображений с Открытым исходным кодом Open CV.
- Отслеживание и распознание объектов с использованием библиотеки Open CV.
- Проводить анализ.
- Сделать выводы о результатах анализа.

**Постановка задачи:**

- Создать программный комплекс для анализа изображения предаваемого с камеры с разрешённою 640x480.
- Проводить детектирования объекта с вероятностью не менее 60%.
- На основе полученных результатов провести анализ для вычисление геометрических параметров исследуемой системой.
- Построить эквивалентную математическую модель с физической модель.
- Произвести сравнения с соответствующий модель.
- Сделать выводы.

**Этапы выполнения работы:**

- Изучение предметной области.
- Выбор инструментария.
- Исследование алгоритма.
- Изучение основных методов и функций библиотеки Open CV.
- Реализация коды программы.
- Анализ полученных результатов.

# 1 Теоретическая часть

## 1.1 Компьютерное зрение

Человек получает большой объём информации с помощью зрения. Для человека возможность видеть и распознавать объекты является естественной и привычной возможностью, которое для компьютера является чрезвычайно сложной задачей.

**Компьютерное зрение** — это теория и технология создания машин, которые могут производить обнаружение, отслеживание и классификацию объектов.

**Цель компьютерного зрения** - это автоматизация процесса распознавания и обработки цифровых изображений и видеофайлов, то есть как научить компьютер распознавать объекты на статическом изображении или в видео.

Компьютерное зрение как научная дисциплина появилась в конце 1970-х годов (в оригинале Computer vision, далее CV) и относится к теории и технологии создания искусственных систем, которые получают информацию из изображений. Видеоданные могут быть представлены множеством форм, таких как видеопоследовательность, изображения с различных камер или трехмерными данными, например, с устройства Kinect или медицинского сканера.

На вход компьютеру подаётся некоторое изображение, а на выходе возможно получение границ всех важных объектов и их классификацию. Входные данные могут содержать некоторую контекстную информацию, например, "камера установлена в машине". Решением может быть "есть ли человек в этой сцене". Новым представлением может быть процесс превращения цветного изображения в чёрно-белое или устранение эффекта движения камеры из последовательности изображений.

Как технологическая дисциплина, компьютерное зрение стремится

применить теории и модели компьютерного зрения к созданию систем компьютерного зрения.

Примерами применения таких систем могут быть:

1. Системы управления процессами (промышленные роботы, автономные транспортные средства).
2. Системы видеонаблюдения.
3. Системы организации информации (например, для индексации баз данных изображений).
4. Системы моделирования объектов или окружающей среды (анализ медицинских изображений, топографическое моделирование).
5. Системы взаимодействия (например, устройства ввода для системы человеко-машинного взаимодействия).
6. Системы дополненной реальности.
7. Вычислительная фотография, например, для мобильных устройств с камерами.

Человеческий мозг разделяет сигнал, поступивший от зрения на множество каналов, которые впоследствии передают различного вида информацию в мозг человека. Мозг устроен таким образом, что внимание концентрируется только на важных участках изображения, исключая при этом остальные. Сигналы, поступающие на ассоциативные входы, пришедшие от датчиков контроля мышц и всех других чувств, позволяют мозгу опираться на перекрестные ассоциации, накопленные за годы прожитых лет. За счет обратной связи в мозге, процесс повторяется вновь и вновь и включает в себя датчик (глаз), который механически управляет освещением с помощью радужной оболочки и настраивает прием на поверхности сетчатки.

В системе компьютерного зрения все по-другому. Компьютер получает только сетку с числами от камеры или с диска.

## 1.2 Области применения компьютерного зрения

Применяется компьютерное зрение во многих областях включая промышленные, военные и бытовые системы.

Одним из наиболее важных применений является обработка изображений в медицине. Эта область характеризуется получением информации из видеоданных для постановки медицинского диагноза пациентам. В большинстве случаев, видеоданные получают с помощью микроскопии, рентгенографии, ангиографии, ультразвуковых исследований и томографии. Примером информации, которая может быть получена из таких видеоданных является обнаружение опухолей, атеросклероза или других злокачественных изменений. Также примером может быть измерение размеров органов, кровотока и т. д. Эта прикладная область также способствует медицинским исследованиям, предоставляя новую информацию, например, о строении мозга или качеству медицинского лечения.

Другой прикладной областью компьютерного зрения является *промышленность.* Здесь информацию получают для целей поддержки производственного процесса. Автоматизация позволяет не только удешевить процесс производства, но и уменьшить количество брака на выходе. Примером может служить контроль качества, когда детали или конечный продукт автоматически проверяются на наличие дефектов. Другим примером является измерение положения и ориентации деталей, поднимаемых рукой робота.

Самой большой областью применения компьютерного зрения является *военное применение*. Примерами могут быть обнаружение вражеских солдат и транспортных средств и управление ракетами. Наиболее совершенные системы управления ракетами, основываясь на получаемых видеоданных, могут посылать ракету в заданную область и вместо конкретной цели, производить селекцию целей, когда ракета достигает заданной области. Используется автоматическая обработка данных, которая уменьшает

сложность и увеличивает надежность получаемой информации и способствует принятию стратегических решений.

Новой областью применения компьютерного зрения являются автономные транспортные средства, включая подводные, наземные (роботы, машины), воздушные. Уровень автономности изменяется от полностью автономных (беспилотных) до транспортных средств, где системы, основанные на компьютерном зрении, поддерживают водителя или пилота в различных ситуациях. Полностью автономные (беспилотные) транспортные средства используют компьютерное зрение для навигации, то есть для получения информации о месте своего нахождения, для создания карты окружающей обстановки, для обнаружения препятствий. Также они используются для определённых конкретных задач, например, для обнаружения лесных пожаров. Примерами таких систем могут быть система предупредительной сигнализации о препятствиях на машинах и системы автономной посадки самолетов.

В настоящее время CV получило широкое применение - это распознавание номеров автомобилей, лиц людей, текста, дороги, автомобилей и многое другое.

## 1.3 Задачи компьютерного зрения

**Распознавание –** это классическая задача в компьютерном зрении, обработке изображений и машинном зрении, которая определяет содержат ли видеоданные некоторый характерный объект, особенность или активность. Существующие методы решения этой задачи эффективны только для отдельных объектов, таких как простые геометрические объекты (например, многогранники), человеческие лица, печатные или рукописные символы, автомобили и только в определённых условиях, обычно это определённое освещение, фон и положение объекта относительно камеры.

**Движение** – это задачи, связанные с оценкой движения, в которых последовательность изображений или видеоданных обрабатывается для нахождения оценки скорости каждой точки изображения или 3D сцены. Примерами таких задач являются: определение трехмерного движения камеры или слежение, то есть следование за перемещениями объекта (например, машин или людей).

**Восстановление изображений** – это удаление шума. Например, шум датчика, размытость движущегося объекта и т. д. Решаются эти задачи с помощью различных типов фильтров, таких как фильтры нижних или средних частот. Более сложные методы удаления шумов используют первоначальный анализ видеоданных на наличие различных структур, таких как линии или границы, и только потом управление процессом фильтрации на основе этих данных.

## 1.4 Функции, типичные для многих систем компьютерного зрения

**Получение изображений** - цифровые изображения получаются от одного или нескольких датчиков изображения, которые включают различные типы светочувствительных камер, датчики расстояния, радары, ультразвуковые камеры и т.д. В зависимости от типа датчика полученные данные, могут быть последовательностью изображений, 2D-изображением или 3D-изображением. Интенсивности света обычно соответствуют значения пикселей в одной или нескольких спектральных полосах (цветные или изображения в оттенках серого), которые также зависят от различных физических измерений, как глубина, поглощение или отражение звуковых или электромагнитных волн, ядерный магнитный резонанс.

**Предварительная обработка**: для получения необходимой информации нужно обработать видеоданные так, чтобы они удовлетворяли условиям используемого метода. Например, повторная выборка для

убеждения в том, что координатная система изображения верна, удаление шума с тем, чтобы удалить искажения, вносимые датчиком, улучшение контрастности, для того чтобы нужная информация могла быть обнаружена и масштабирование для лучшего различения структур на изображении.

**Выделение деталей:** детали изображения различного уровня сложности выделяются из видеоданных. Например, линии, границы и кромки или локализованные точки интереса, такие как углы, капли или точки и более сложные детали, которые относятся к структуре, форме или движению.

**Детектирование/Сегментация:** на определённом этапе обработки принимается решение о том, какие точки или участки изображения являются важными для дальнейшей обработки. Например, выделение определённого набора интересующих точек или сегментация одного, или нескольких участков изображения, которые содержат характерный объект.

**Высокоуровневая обработка:** входные данные здесь представляют небольшой набор данных, состоящих из наборов точек или участка изображения, в котором предположительно находится определённый объект. Например, проверка данных,

удовлетворяют ли они условиям, зависящим от метода и применения, оценка характерных параметров, таких как положение или размер объекта, или классификация обнаруженного объекта по различным категориям.

## 1.5 Проблемы обнаружения объектов в OpenCV

**Ракурсы.** При сравнении двух фотографий лица одного человека в фас и в профиль мы увидим, что они имеют больше различий, чем сходств. То же самое происходит и с множеством других объектов: с одного ракурса мы можем определить объект ухо, а с другого- уже нет. Но если знать, что объекты всегда будут изображены в определённой позиции, то задача определения данных объектов значительно облегчается. С другой стороны,

при изменении ракурса, сложность определения объекта возрастает в много раз.

**Освещение.** Человек может получить множество информации с помощью освещения и теням от объектов и также с использованием контекста. Тени и освещение добавляют дополнительные детали. Например, цвет одежды объекта зависит от освещения. Обучение машины пониманию теней и изменению из-за них цвета объекта является одной из задач CV.

**Масштаб.** Объекты могут быть различных размеров в зависимости от их расположения. Человеческий глаз может догадаться, что на одном изображении находится автомобиль, а на втором — игрушечная машинка. Компьютер же определит эти объекты одинаково, если его не научить.

**Деформация.** Живые объекты при движении могут претерпевать деформации. Например, после аварии автомобиль может сильно измениться и распознать его станет сложнее.

**Перекрытие**. Например, если две машины едут по разным рядам и одна начнёт обгонять вторую, то первая машина перекроет вторую относительно наблюдателя, который предположим находится на обочине. В результате часть, а иногда вся информации об объекте пропадает.

**Движение.** Многие объекты в движении могут оказаться размытыми или просто изменить свой ракурс.

**Маскировка**. Для безопасности объекта человек приходиться использовать маскировку. Например, найти ткань одинакового цвета со стеной монотонного цвета не составит большого труда для злоумышленника, но запросто обманет примитивную автоматизированную систему видеонаблюдения

**Внутриклассовая изменчивость**. Все люди выглядят по-разному: у каждого своего цвета лица, форма головы, ушей и т.д. У некоторых людей может не быть определённых частей тела (рук, ног). Многие неодушевлённые объекты, такие как камни могут отличаться от объектов того же типа. Это касается и вещей, произведённых самими людьми.

**Контекст.** Из контекста человек получает большой объём информации. Например, медведь находящийся в музее без решётки является чучелом и опасности не представляет. Но если он в диком лесу, то он живой и опасен для людей.

# 2 Методы нахождения объектов на изображении

Основной задачей, возникающей при отслеживании объектов в видеопотоке, является их автоматическое нахождение на каждом отдельно взятом кадре.

## 2.1 Сегментация изображений (IMAGE SEGMENTATION)

**Сегментация** – это процесс разделения цифрового изображения на несколько множеств пикселей, или присвоения каждому пикселю таких меток, чтобы пиксели с одинаковыми метками имели общие визуальные характеристики.

Цель сегментации состоит в упрощении представления изображения, чтобы его было легче анализировать. В результате сегментации обычно выделяют границы и объекты на изображениях. Пиксели, принадлежащие к одним и тем же сегментам, похожи по некоторой вычисленной характеристике. Например, похожи по цвету или яркости, а соседние элементы значительно различаются по ней (рисунок 2.1).

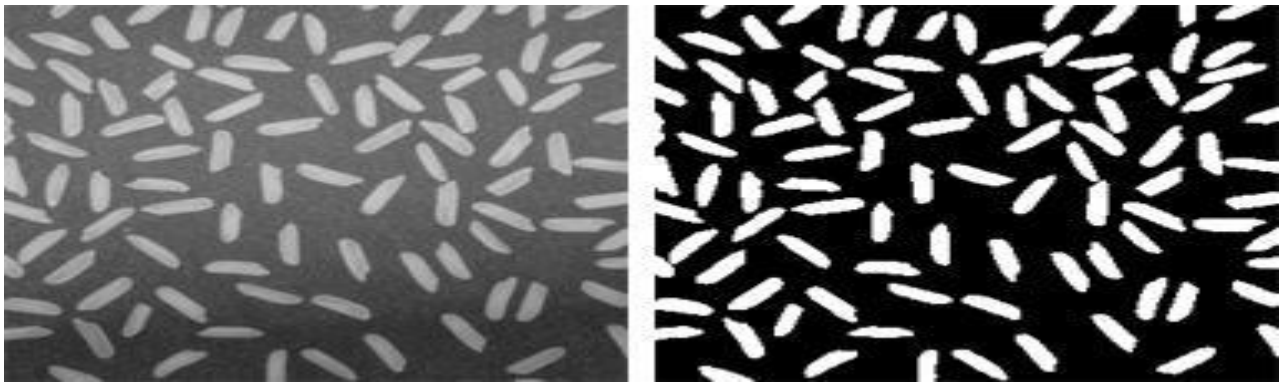

Рисунок 2.1- Пример сегментации изображения

Достоинства метода:

- Простота использования в тех ситуациях, когда исследуемые объекты значительно отличаются от остального фона по какому-либо параметру, например, в биоинформатике или поиск дефектов материалов.

Недостатки:

- Низкая универсальность методов.
- Необходимость тонкой настройки параметров работы алгоритма распознавания в каждом отдельном случае.
- Высокая чувствительность к изменениям освещения сцены.

## 2.2 Сопоставление шаблонов (TEMPLATE MATCHING)

Идея поиска объектов на изображении по шаблону заключается в сравнении шаблона, на котором изображен искомый объект, с подобластями обрабатываемого изображения. Метод хорошо работает с чёрно-белыми, контрастными изображениями. При его использовании необходимо учитывать такие моменты как освещенность, возможность объекта изменять своё положение относительно камеры и другие особенности, которые могут изменить полученные изображения.

При использовании данного метода необходимо:

1. Зафиксировать объект.
2. Для определения лица человека на изображении, лицо должно быть всегда в фас.
3. Изображение должно быть качественным, то есть отсутствие шумов и т.п.
4. Необходимо получить шаблон.

Шаблон - это изображение объекта, которое нужно найти на другом изображении производится сравнение интенсивности. Шаблон попиксельно

сравнивается с изображением, на котором необходимо найти данный объект и считается, что объект найден если он соответствует заданному порогу.

Существует несколько метрик, позволяющих определить, насколько качественно получилось найти изображение:

**Сумма абсолютных разностей** - SAD (Sum of absolute differences)

$$SAD(x,y) = \sum_x \sum_y |I_1(x,y) - I_2(x,y)| \tag{2.1}$$

$I_1$ - интенсивность пикселя в шаблоне.

$I_2$ - интенсивность пикселя в изображении.

$(x,y)$ - координаты пикселя.

При наименьшем SAD достигается наилучший результат.

**Сумма квадратов разностей** - SSD (Sum of squared differences)

$$SAD(x,y) = \sum_x \sum_y \left(I_1(x,y) - I_2(x,y)\right)^2 \tag{2.2}$$

При SSD желательно получить результат близкий к нулю.

**Произведение** - CC (Cross-correlation)

$$CC(x,y) = \left(I_1(x,y) * I_2(x,y)\right) \tag{2.3}$$

При CC наилучший результат должен равняться единице.

Достоинства:

- Хорошая применимость при анализе сцен, в которых камера статична, и все экземпляры искомых объектов выглядят одинаково. Например, при распознавании деталей на сборочной линии.

Недостатки:

- Неустойчивая работа в случае масштабирования, сдвига, поворота

изображений, а также в случаях, когда распознаваемый объект виден не полностью.

## 2.3 Выделение краёв (EDGE DETECTION)

Выделение краёв, границ или контуров является следующим методом нахождения объектов на изображении. Благодаря краям объекта, можно попытаться понять, что за объект представлен на изображении.

Края можно находить по резким изменениям цвета поверхности, глубины и нормали поверхности. Когда края будут найдены, далее для нахождения отдельных объектов на изображении уже используется метод сопоставление шаблонов.


Рисунок 2.2 - Поиск краёв

Из рисунка 2.2 видно, что край - это место резкого изменения значений функции интенсивности изображений. Для нахождения края, берем функцию интенсивности изображения и находим точки экстремума первой производной, которые будут соответствовать границам изображения. Найти границу изображения можно также с помощью градиента.

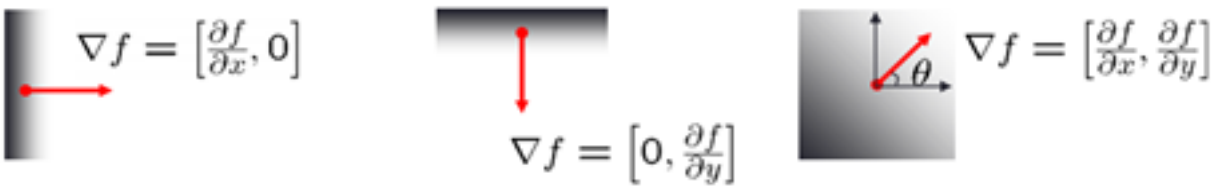


Рисунок 2.3 - Иллюстрация работы градиента.

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right] \quad (2.4)$$

Из рисунка 2.3 видно, что градиент направлен в сторону наибольшего изменения интенсивности.

Направление градиента задаётся как:

$$\theta = \tan^{-1}\left(\frac{\left(\frac{\partial f}{\partial x}\right)}{\left(\frac{\partial f}{\partial y}\right)}\right) \quad (2.5)$$

Сила края задаётся величиной или нормой градиента:

$$|\nabla f| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \quad (2.6)$$

Этот метод при большом количестве зашумлений будет работать плохо и для улучшения и применения используют сглаживание и фильтрацию.

Самые известные фильтры:

Робертса $\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$ $\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$

Превитт

$$\begin{bmatrix}-1 & -1 & -1\\ 0 & 0 & 0\\ 1 & 1 & 1\end{bmatrix} \begin{bmatrix}-1 & 0 & 1\\ -1 & 0 & 1\\ -1 & 0 & 1\end{bmatrix}$$

Собеля

$$\begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{bmatrix} \begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}$$

Недостатки метода вычисления градиента:

- Толщина - полученные края получаются толстые, размытые.
- Непонятность - какие линии соотносятся друг с другом и где именно находятся линии одного объекта.

## 2.4 Детектор границ канни (CANNY EDGE DETECTOR)

**Края (границы)** - это такие кривые на изображении, вдоль которых происходит резкое изменение яркости или других видов неоднородностей. Края отражают важные особенности изображения. Причинами возникновения краёв являются изменение освещенности, изменение цвета и изменение глубины сцены (ориентации поверхности). Выделение существенных характеристик изображения и сокращение объема информации для последующего анализа являются главными целями преобразования изображения в набор кривых

В 1986 году Джоном Канни был разработан детектор границ, который стал самым популярным методом выделения границ. Канни изучил математическую проблему получения фильтра, оптимального по критериям выделения, локализации и минимизации нескольких откликов одного края. Детектор Канни должен реагировать на границы и точно определять линию границы, игнорировав ложные границы. На каждую границу реагировать только один раз, что позволило избежать восприятия широких полос изменения яркости как совокупности границ.

Пикселями границ объявляются точки, в которых достигается локальный максимум градиента в направлении вектора градиента.

Алгоритм работы этого метода следующий:

1. Сглаживание. Производится размытие изображения, чтобы избавиться от шумов.
2. Поиск градиентов. Рассчитываются градиенты изображения: сила и направление. В местах максимального значения градиента — граница.
3. Выделение локальных максимумов. Полосы в несколько пикселей уменьшают до полос в один пиксель.
4. Двойная пороговая фильтрация. Потенциальные границы определяются двумя порогами (верхний и нижний). Верхний порог необходим для инициализации кривых, нижний — для продолжения.

## 2.5 Примитивы хаара (HAAR-LIKE FEATURES)

Примитивы Хаара представляют собой прямоугольники, состоящие из смежных областей (рисунки 2.4 и 2.5).

Примитивы Хаара представляют собой набор пикселей, часть из которых белые, а часть чёрные. Это первый алгоритм детектирования лиц, работающий в реальном времени.

На рисунке 2.6 мы видим пример нахождения признаков Хаара на изображении.

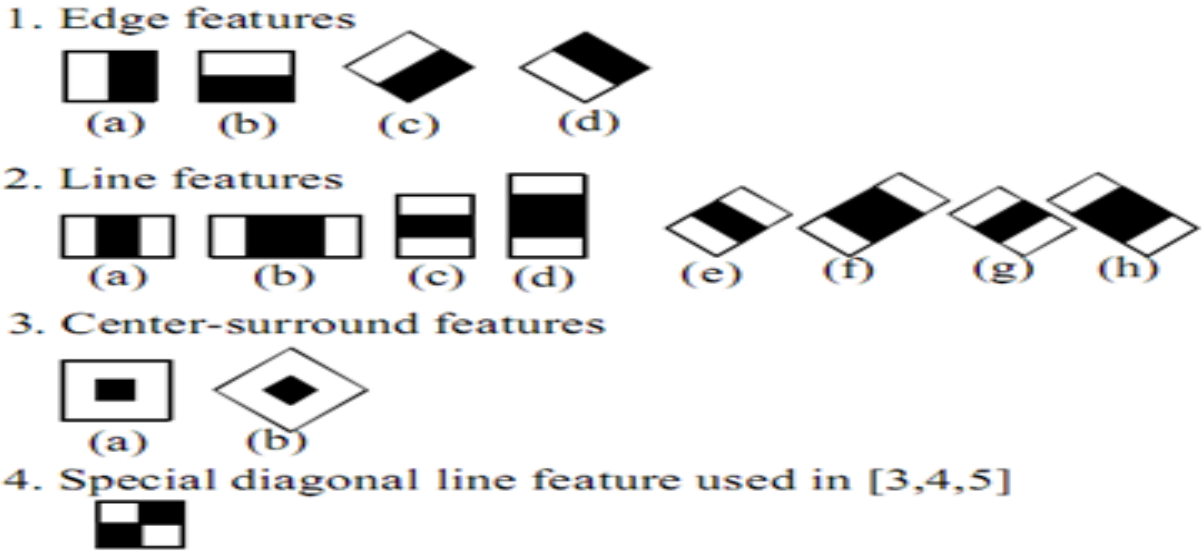


Рисунок 2.4 -Примитивы Хаара.

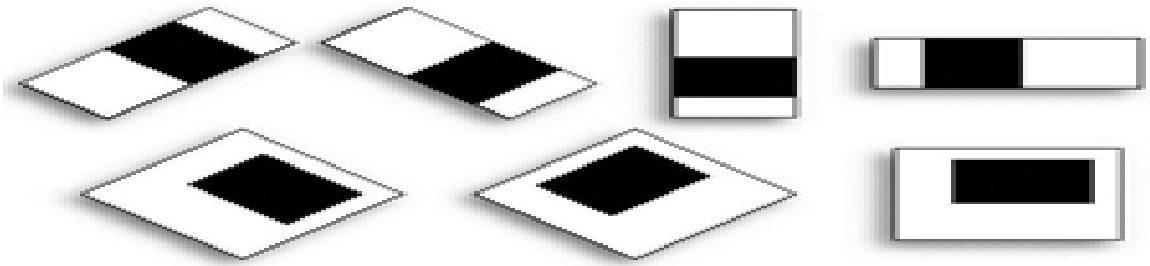

Рисунок 2.5 - Дополнительные примитивы Хаара, используемые в Open

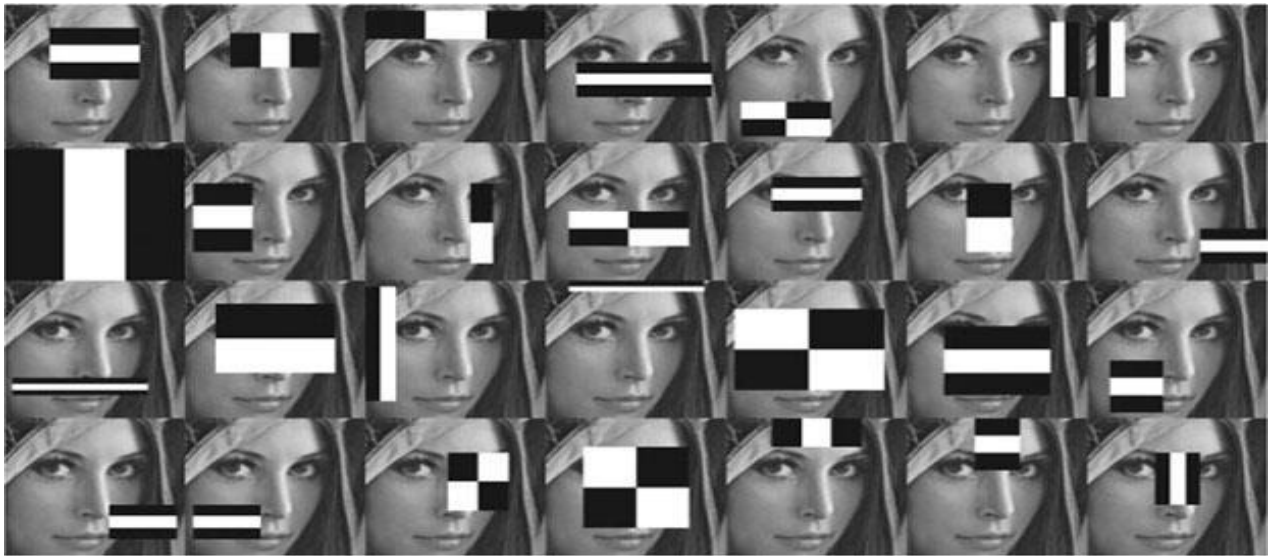

Рисунок 2.6 - Пример нахождения признаков Хаара на изображении

Примитивы Хаара представляют собой набор пикселей, часть из которых белые, а часть чёрные.

Для сравнения примитивов с изображением используется интегральное представление изображений. На основе рабочего изображения создаётся матрица, которая должна будет совпадать по размерам с исходным изображением. В элементах этой матрицы хранятся суммы интенсивностей всех пикселей, находящихся левее и выше данного элемента.

В виде формулы это выглядит следующим образом:

$$L(x,y) = \sum_{i=0,j=0}^{i\le x, j\le y} I(i,j) \qquad (2.7)$$

Где $(i,j)$ - яркость пикселя исходного изображения.

Полученная новая матрица $L(x,y)$ хранит в себе сумму интенсивностей пикселей в прямоугольнике от $(0,0)$ до $(x,y)$ причём значение элемента матрицы с координатой $(x,y)$ будет максимальным и является суммой всех предыдущих элементов матрицы. Следовательно, расчёт новой матрицы занимает линейное время, пропорциональное числу пикселей в изображении, поэтому интегральное изображение просчитывается за один проход. Расчёт матрицы осуществляется по следующей формуле:

$$L(x,y) = I(x,y) - L(x-1,y-1) + L(x,y-1) + L(x-1,y) \qquad (2.8)$$

### 2.5.1 Математическое определение примитивов Хаара

Признак – это отображение $f: \quad x \Longrightarrow Df$ , где $Df$ - множество допустимых значений признака.

Если заданы признаки $f_1, \ldots . f_n$ , то вектор признаков $f_1(x), \ldots . f_n(x)$ называется признаковым описанием объекта $x \in X$.

Признаковые описания допустимо отождествлять с самими объектами.

При этом множество $X = Df_1 * \ldots \ldots * Df_n$ называют признаковым пространством.

Признаки делятся на следующие типы в зависимости от множества $Df$ :

1. Бинарный признак, $Df$ = {0,1};
2. Номинальный признак: $Df$ = конечное множество;
3. Порядковый признак: $Df$ = конечное упорядоченное множество;
4. Количественный признак: $Df$ = множество действительных чисел.

Вычисляемым значением такого признака будет:

$$F = X - Y \qquad (2.9)$$

где $X$ – сумма значений яркостей точек, закрываемых светлой частью признака,

$Y$ – сумма значений яркостей точек, закрываемых тёмной частью признака.

Для их вычисления используется понятие интегрального изображения. Признаки Хаара дают точечное значение перепада яркости по оси $X$ и $Y$, соответственно.

Достоинством признаков Хаара является сравнительно высокая скорость вычисления.

## 2.6 Поиск объекта по цвету

**Цветовое пространство** - это модель представления цвета, основанная на использовании цветовых координат, то есть любой цвет будет представлен точкой, имеющей определённые координаты. Для хранения цифровых изображений используется обычно цветовое пространство RGB.

В RGB каждой из трех осей или каналов присваивается свой цвет: красный, зеленый и синий. На каждый канал выделяется по 8 бит информации и соответственно интенсивность цвета на каждой оси будет принимать значения в диапазоне от 0 до 255. Все цвета в цифровом пространстве RGB получаются путем смешивания трех основных

цветов: красный, зеленый синий.

На рисунке 2.7 мы видим цветное изображение, представленное в виде 3 матриц: **red, green, blue**. Каждая небольшая коробка представляет собой пиксель изображения. Эксперименты показывают, что RGB не всегда хорошо подходит для анализа информации, так как геометрическая близость цветов достаточно далека от того, как человек воспринимает близость тех или иных цветов друг к другу в реальных изображениях, эти пиксели настолько малы, что человеческий глаз не может различать их.

Существуют и другие цветовые пространства.

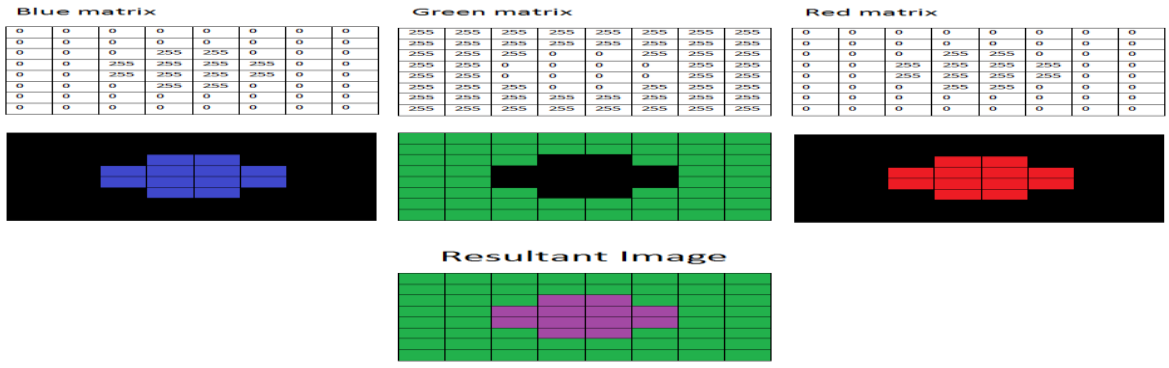


Рисунок 2.7 - Цветное изображение представлено с использованием 3 матрицы

Существуют и другие цветовые пространства. Наиболее подходящим цветовым пространством для сегментации изображения на основе цвета является цветовое пространство HSV.

**Пространство HSV** (англ. Hue, Saturation, Value- тон, насыщенность, значение) —это цветовая модель, в которой координатами цвета являются:

**Ось *Value (значение цвета)*** - это яркость и обозначает количество света. Варьируется пределах 0—100 и 0—1. В отличие от RGB на него выделен отдельный канал и его значение не нужно вычислять каждый раз. Это черно-белая версия изображения и с ней уже можно работать.

**Ось *Hue*** - это цветовой тон и представляется в виде угла. Варьируется в пределах 0-360°,

**Ось *Saturation*** – это насыщенность цвета и зависит от расстояния от центра к краю.

Варьируется в пределах 0-100 или 0-1. Чем больше этот параметр, тем «чище» цвет, поэтому этот параметр иногда называют чистотой цвета. А чем ближе этот параметр к нулю, тем ближе цвет к нейтральному серому (рисунок 2.8).

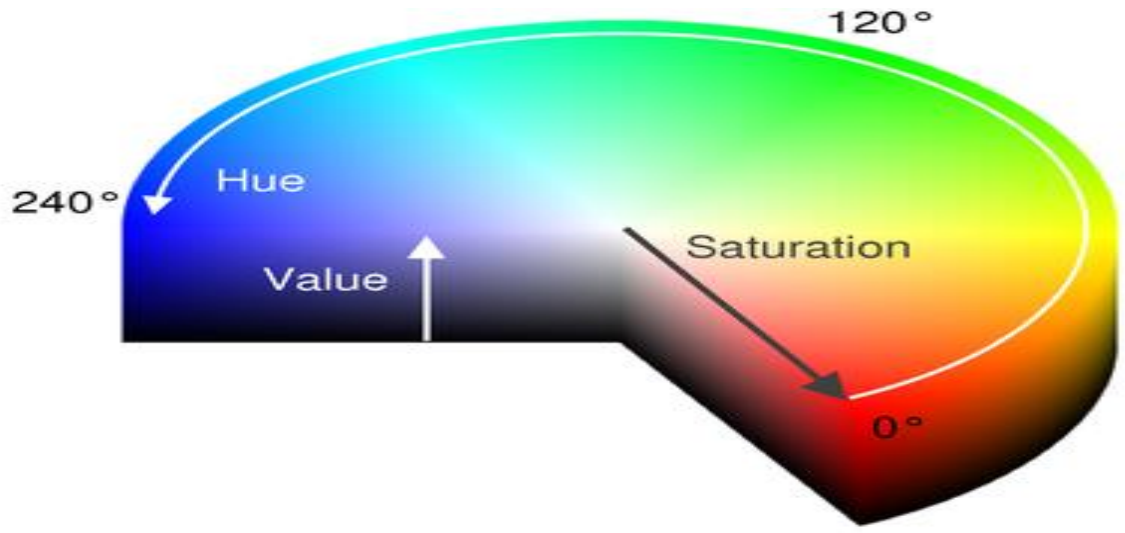


Рисунок 2.8 - Цветовое пространство HSV

Пространство HSV намного ближе к тому, как мы представляем себе

цвета. Например, если показать человеку в темноте красный и зеленый объект, то он не сможет различить цвета.

В HSV происходит то же самое. Чем ниже по оси V мы продвигаемся, тем меньше становится разница между оттенками, так как снижается диапазон значений насыщенности. На схеме это выглядит как конус, на вершине которого предельно черная точка (рисунок 2.9)

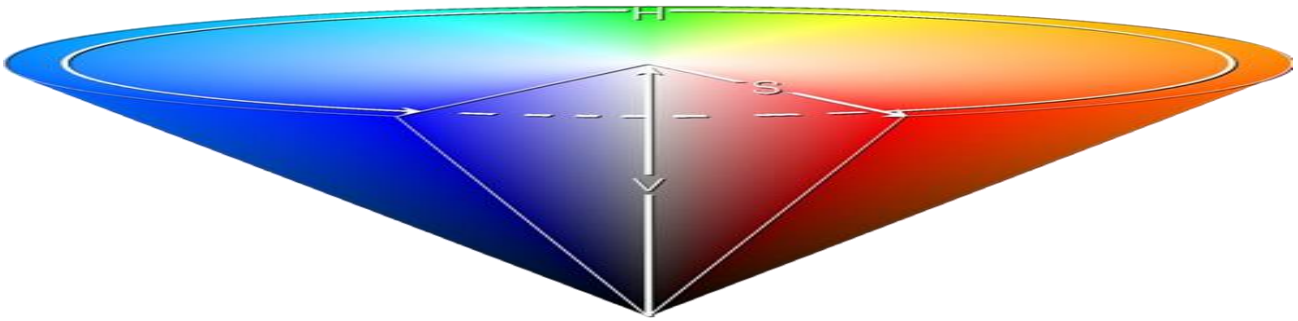


Рисунок 2.9 - Коническое представление модели HSV

В основе этого метода лежит идея фильтрации картинки по цвету. Этот метод можно применять, когда объект существенно отличается по цвету от фона и условия освещения изменяются мало.

**Метод поиска объекта по цвету** является одним из самых распространённых методов. В основе этого метода лежит идея фильтрации картинки по цвету. Успешное применение этого метода зависит от объекта, который должен отличается по цвету от основного фона и условий освещения, которые мало изменяются.

Цвет - это свойство тел отражать или испускать видимое излучение определенного спектрального состава и интенсивности. Например, желтый цвет один из самых ярких и последний цвет, которые видят слабовидящие люди. Поэтому для них приклеивают жёлтые круги на дверях магазинов, чтобы они различали стеклянную дверь, покрываются жёлтой краской бордюры и полосы на ступеньках пешеходных переходов.

Видимый цвет является результатом взаимодействия спектра излучаемого света и поверхности. На поиск объекта по цвету играют множество факторов. Освещенность –это один из факторов, влияющих на поиск объекта по цвету. Например, если мы будем освещать белый лист светом красной лампочки, то лист будет казаться нам красным.

Поэтому для поиска объекта с таким цветом нужны следующие условия:

- целевая область должна заранее выделяется в цветовом пространстве и реализоваться только в случае поиска объектов заранее известных цветов.
- целевая область должна определяется указаниями пользователя и описываться цветовыми терминами естественного языка человека. Например, синий, красный, зеленый, ярко-желтый, и т.д.

# 3 Выбор инструментов

В данной работе были выбраны язык программирования C++, Microsoft Visual Studio Community 2013 и Microsoft Visual Studio Community 2015 и библиотека компьютерного зрения с открытым исходным кодом версия Open CV 2.4.10 и версия Open CV 3.1.1.

## 3.1 Язык программирования C++

Разработчиком языка C++ является Бьерн Страуструп. Основные работы велись в исследовательском центре компании Bell Labs. Предшественник C++ – это Язык Си с классами который появился в 1979 году. В 1997 году был принят международный стандарт C++, которое обеспечило единообразие всех реализаций языка C++. Не менее важным результатом стандартизации стало то, что в процессе выработки и утверждения стандарта язык был уточнен и дополнен рядом существенных возможностей.

Язык C++ является универсальным языком программирования, в дополнение к которому разработан набор разнообразных библиотек.

C++ широко используется в системном программировании. На нем можно писать высокоэффективные программы, в том числе операционные системы (ОС), драйверы и т.п. Язык C++ – один из основных языков разработки трансляторов.

Обработка сложных структур данных – текста, бизнес-информации, веб-страниц и т.п. – одна из наиболее распространенных возможностей применения языка.

Язык C++ в настоящее время является одним из наиболее распространенных языков программирования в мире.

## 3.2 MATLAB

MATLAB является высокоуровневым языком программирования и интерактивной средой программирования. Это мощная система для математических вычислений, их визуализации, для обработки сигналов, финансового инжиниринга и многого другого.

## 3.3 Открытая библиотека OpenCV

Библиотека Open CV (Open Source Computer Vision Library) - это библиотека компьютерного зрения с открытым исходным кодом, то есть с набором алгоритмов, реализованных в виде функций CV, написанная на C и C++.

Главное достоинство данной библиотеки в том, что она полностью открытая и бесплатная как в учебных, так и в коммерческих целях. Вначале разработкой библиотеки занималось русское отделение Intel в Нижнем Новгороде (до релиза).

Альфа версия вышла в январе 1999 года, а первый релиз — в 2006 году. В данный момент последняя стабильная версия 2.4.9, также выпущена альфа-версия 3.0.

Существуют дистрибутивы под Linux, Windows, Mac, iOS и Android. Библиотека имеет онлайн-документацию и поддерживает следующие языки программирования: C/C++, Python, Java/Scala MATLAB. Может свободно использоваться в академических и коммерческих целях — распространяется в условиях лицензии BSD.

Open CV написана на языке высокого уровня (C/C++) и содержит алгоритмы для: интерпретации изображений, калибровки камеры по эталону, устранение оптических искажений, определение сходства, анализ перемещения объекта, определение формы объекта и слежение за объектом,

3D-реконструкция, сегментация объекта, распознавание жестов и т.д.

Вначале библиотека состояла из 5 основных модулей: Excore, Highqui, Cvaux,CvCam, но начиная с версии 2.2, структура библиотеки изменилась и на данный момент выглядит следующим образом:

- **opencv_core** - ядро: базовые структуры, вычисления: математические функции, генерация псевдослучайных чисел, DFT, DCT, ввод и вывод в XML и т.п.
- **opencv_imgproc** - обработка изображений: фильтры,преобразования т.д.
- **opencv_highgui** - простой UI, загрузка и сохранение видео и изображений.
- **opencv_ml**- методы и модели машинного обучения: SVM, деревья принятия решений и т.д.
- **opencv_features2d** - различные дескрипторы: SURF.
- **opencv_video -** анализ движения и отслеживание объектов: оптический поток, шаблоны движения, устранение фона.
- **opencv_objdetect** - детектирование объектов на изображении: вейвлеты Хаара, HOG и т. д.
- **opencv_calib3d** - калибровка камеры, поиск стерео-соответствия и элементы обработки трёхмерных данных.
- **opencv_flann** - библиотека быстрого поиска ближайших соседей: FLANN.
- **opencv_contrib** - сопутствующий код, ещё неготовый для применения.
- **opencv_legacy** - устаревший код, сохранённый ради обратной совместимости.
- opencv_gpu - ускорение некоторых функций Open CV за счёт CUDA(Nvidia).

### 3.3.1 Работа с видео

В OpenCV есть несколько функций для работы с видео, наиболее

используемыми из которых являются функция чтения и записи видео.

CvCapture – структура, содержащая необходимую информацию для чтения кадров из камеры или из видеопотока. Функции, которые используются в зависимости от источника (листинг3.1).

Листинг 3.1 – Прототипы чтения видеопотока.

```
CvCapture*
cvCreateFileCapture(
const char* filename
);



CvCapture*
cvCreateCameraCapture( int
index
);
```

**cvCreateFileCapture()** принимает всего один аргумент – имя AVI или MPG-файла. После вызова этой функции Open CV открывает и подготавливает файл для чтения.

Если открытие файла произошло успешно, то функция вернет указатель на структуру CvCapture. Если по какой-то причине, файл не существует или не найден соответствующий кодек, функция вернет NULL. Необходимо знать какой кодек применять и установлен ли он на компьютере.

Например, необходимо прочитать файл, закодированный с помощью DIVX или сжат с помощью MPG4 на компьютере с ОС Windows. Для корректной обработки должна быть установлена соответствующая библиотека, которая предоставит все необходимые ресурсы для декодирования видео. Необходимо проверять возвращаемое значение на

NULL, так как все компьютеры имеют разный набор ПО (например, может отсутствовать библиотека для декодирования видео).

Функция cvCreateCameraCapture() принимает идентификатор, указывающий к какой камере необходимо получить доступ. Если камера одна, то идентификатор можно установить в 0.

index– это сумма из порядкового номера и "домена". "Домен" указывает на тип камеры.

### 3.3.2 Обработка изображения в OpenCV

Open CV предоставляет быстрый и простой в использовании интерфейс для выполнения морфологических преобразований над изображениями.

Основные морфологические преобразования – расширение и размытие (сужение) Они применяются:

- Для удаления шума
- Выделение отдельных элементов
- Соединение разнородных элементов на изображении.
- Для поиска неровностей интенсивности или отверстий
- Градиента изображения.

**Расширение** – свёртка изображения (или области изображения), которое именуется A, с некоторым ядром, которое именуется B. Ядро может быть любой формы и размера, но обычно это небольшой сплошной квадрат или диск с якорем в центре и имеет одну определенную точку привязки (якорь). Ядро можно рассматривать в качестве шаблона или маски и его эффект на расширение зависит от оператора локального максимума.

При сканировании изображения ядром B происходит вычисление локального максимума пикселя, перекрываемого B, и затем значение пикселя, лежащего под опорной точкой, заменяется этим максимальным значением. Это приводит к появлению ярких областей на изображении;

схематично этот рост показан на рисунке 3.1. Этот рост именуется "расширением оператора".

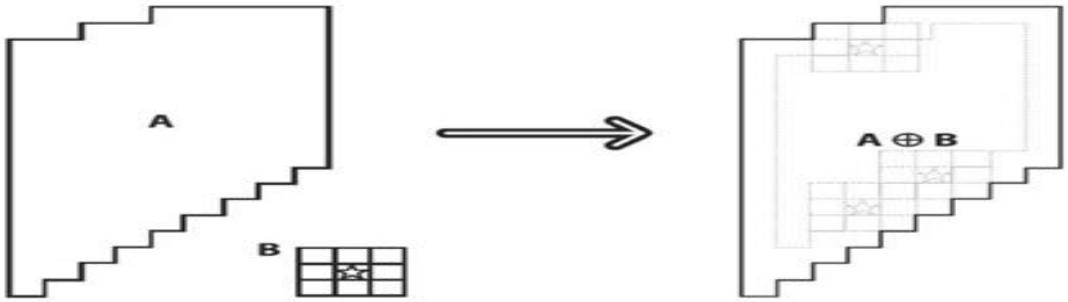


Рисунок 3.1 – Морфологическое преобразование – расширение

**Размытие** – обратная операция. Действие оператора размытия заключается в вычислении локального минимума под ядром. Данный оператор создаёт новое изображение на основе исходного по следующему алгоритму:

При сканировании изображения ядром B происходит вычисление локального минимума пикселя, перекрываемого B, и затем значение пикселя, лежащего под опорной точкой, заменяется этим минимальным значением. Схематично сужение показано на рисунке 3.2

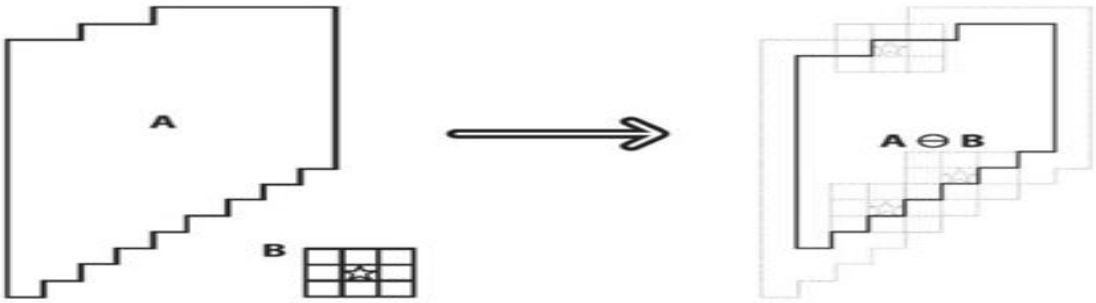


Рисунок 3.2 - Морфологическое преобразование – размытие

Морфологические преобразования выполняются над бинарными изображениями, получаемыми с помощью порогового преобразования. Расширение увеличивает область A, а размытие уменьшает область A.

Расширение применяется для того, чтобы сгладить вкрапления, а размытие, чтобы сгладить выступы. Конечный результат всегда будет зависеть от ядра.

В Open CV для выполнения этих преобразований используются функции **cvErode()** и **cvDilate()** ( листинг 3.2).

Листинг 3.2 – Прототипы функций **cvErode() и cvDilate().**

```
void
cvErode(
IplImage*
src ,
IplImage*
dst ,
IplConvKernel* B =
NULL, int
iterations = 1
);

void
cvDilate(
IplImage*
src ,
IplImage*
dst ,
IplConvKernel* B =
NULL, int
iterations = 1
);
```

- Обе функции допускают использование одного и того же изображения в качестве исходного и конечного.

- Третий аргумент – ядро, которое по умолчанию равно NULL. Если ядро равно NULL, то используется ядро 3x3 с якорем в центре.

- Четвертый аргумент - количество итераций. Если указанное значение не равно 1, то операция будет применяться несколько раз в течение одного вызова функции.

Операция размытия используется для устранения "вкраплений" шума на изображении. Суть операции сужения в том, что вкрапления и шумы размываются, в то время как большие и соответственно более значимые регионы не затрагиваются.

Операция расширения используется при попытке найти связанные компоненты, то есть большие отдельные области аналогичного цвета или интенсивности. Полезность расширения возникает во многом из-за того, что, большая область разбита на несколько более мелкие части шумами, тенями или какими-то аналогичными эффектами. Применение небольшого растягивания приводит к тому, что эти области "плавятся" в одну.

### 3.3.3 Пороговое преобразование

После обработки изображения нужно, чтобы в конечном изображение остались только те пиксели, которые выше или ниже определенного значения.

В Open CV для решения данной проблемы существует функция **cvThreshold()** ( листинг3.3)

Основная идея алгоритма, заложенного в функцию- это попадание в конечный массив только тех пикселей, которые выше или ниже определенного значения.

Листинг 3.3 – Прототип функции **cvThreshold.**

```
double
cvThreshold(
CvArr* src,
CvArr* dst,
double threshold,
double max_value,
int
threshold_type );
```

**Параметры функции:**

1. **src** - исходное изображение (одноканальное, 8-битное или 32-битное);
2. **dst** - целевой массив, должен иметь тот же тип, что и **src** или 8-битный;
3. **threshold** - пороговая величина;
4. **max_value** - максимальное значение;

5 **threshold_type -** тип порогового преобразования.

### 3.3.4 Фильтрация по цвету

Функция **inRange** (листинг 3.4) преобразует цветную картинку в черно-белую маску.

В этой маске, все пиксели, попадающие в заданный диапазон, становятся белыми. Остальные становятся черными.

Листинг3.4 - Прототип функции **cvInRange().**

```
void cvInRange(
const CvArr* src,
const CvArr* lower,
const CvArr* upper,
CvArr* dst)
```

Каждый пиксель **src** сравнивается с соответствующими значениями изображений **lower и upper**. Если значение **src** больше или равно значению **lower** и меньше, чем upper, тогда соответствующее значение в **dst** устанавливается в **0xff**, иначе в 0**.**

### 3.3.5 Контуры

Контур — это внешние очертания (обвод) предмета или объекта. Контурный анализ - это один из важных и очень полезных методов описания, хранения, распознавания, сравнения и поиска графических образов или объектов.

При проведении контурного анализа необходимо чтобы контур содержал достаточную информацию о форме объекта. Во время анализа

внутренние точки объекта во внимание не принимаются.

Проблемы, связанные с выделением контура на изображениях:

- Из-за одинаковой яркости с фоном объект может не иметь чёткой границы или может быть зашумлён помехами, что приводит к невозможности выделения контура;

- Перекрытие объектов или их группировка приводит к тому, что контур выделяется неправильно и не соответствует границе объекта.

Контурный анализ имеет довольно слабую устойчивость к помехам, и любое пересечение или лишь частичная видимость объекта приводит либо к невозможности детектирования, либо к ложным срабатываниям.

Простота и быстродействие контурного анализа, позволяют вполне успешно применять данный подход (при чётко выраженном объекте на контрастном фоне и отсутствии помех).

Контур объекта - это совокупность пикселей, составляющих границу объекта. Для его получения используются детектор границ Канни и методы получения двоичного изображения: пороговое преобразование; выделение объекта по цвету.

Библиотека Open CV реализует удобные методы для детектирования и манипуляции с контурами изображения. Для поиска контуров используется функция **cvFindContours()** (листинг 3. 5).

Листинг 3.5 - Прототип функции **cvFindContours().**

```
cvFindCont
ours(
CvArr*
image,
   CvMemStorage*
storage, CvSeq**
  first_contour,
int header_size
CV_DEFAULT(sizeof(CvContour)), int
mode CV_DEFAULT(CV_RETR_LIST),
int method
CV_DEFAULT(CV_CHAIN_APPROX_SIMPLE),
```

```
CvPoint offset
CV_DEFAULT(cvPoint(0,0)));
```

**Параметры функции**:

1. **image** - исходное 8-битное одноканальное изображение. Ненулевые пиксели

   обрабатываются как **1, а** нулевые как **0.** Для получения изображения из градаций серого можно использовать функции **cvThreshold()** или **cvCanny().**
2. **storage** - хранилище памяти для хранения данных найденных контуров.
3. **first_contour** - указатель, который будет указывать на первый элемент последовательности, содержащей данные найденных контуров.
4. **header_size** - размер заголовка элемента последовательности.
5. **mode** - режим поиска (листинг 3.6).
6. **method** - метод аппроксимации (листинг3. 7).
7. **Offset** - смещение, на которое сдвигать точки контура.

Контуры извлекаются из интересующей области изображения и затем анализируются в контексте целого изображения.

Листинг 3.6 - Режимы поиска.

```
#define CV_RETR_EXTERNAL 0 // найти только
крайние внешние контуры
#define CV_RETR_LIST 1 // найти все контуры и
разместить их списком
#define CV_RETR_CCOMP 2 // найти все контуры и
разместить их в виде 2-уровневой иерархии
#define CV_RETR_TREE 3 // найти все контуры и
разместить их в иерархии вложенных контуров
```

Листинг 3.7 - Методы аппроксимации.

```
#define CV_CHAIN_CODE 0 // цепной код
Фридмана #define CV_CHAIN_APPROX_NONE 1 // все
```

```
точки цепного кода переводятся в точки
#define CV_CHAIN_APPROX_SIMPLE 2     //     сжимает
горизонтальные,  вертикальные   и   диагональные
сегменты  и  оставляет  только  их конечные точки
#define CV_CHAIN_APPROX_TC89_L1   3 //
применяется алгоритм #define
CV_CHAIN_APPROX_TC89_KCOS   4 // аппроксимации
Teh-Chin #define CV_LINK_RUNS
// алгоритм только для CV_RETR_LIST
```

Следующий шаг - это нужно сравнить контуры. Для этого используем вычисление моментов контура. Момент – это грубая характеристика контура, которая вычисляется путем интегрирования (или суммирования) по всем пикселям контура.

Момент контура $(p,\ q)$ определяется следующим образом:

$$(p,q) = \sum_{i=1}^{n} I(x,y)x^p\ y^q \qquad (3.1)$$

где p порядок x, q порядок y, а термин порядок означает степень, в которую возводится компонента суммы.

Суммирование ведется по всем пикселям границы контура и обозначено в уравнение как n.

Если p и q равны 0, то момент равен длине пикселей контура

Листинг 3.8 – Пример использования функции moments ().

Moments moment = moments((cv::Mat)contours[index]);

# 4 Практическая часть. Отслеживание объектов с помощью библиотеки OpenCV

## 4.1 Отслеживание объекта в цветовом пространстве HSV

### 4.1.1 Реализация алгоритма

В области компьютерного зрения важное место занимает проблема поиска и слежения за объектами на основе их цветовых характеристик. К основным преимуществам использования цвета в качестве ключевого признака относятся быстрота выделения объекта и малая изменчивость от угла зрения.

Однако можно выделить две проблемы, затрудняющие использование цвета для поиска объекта:

- Проблема изменения интенсивности освещения и смещения оттенков за счет источника освещения;
- Проблема интерпретации при именовании цветов (color naming), возникающая в случае команд и запросов человека, содержащих указания на тот или иной цвет.

Учитывая перечисленные проблемы, была поставлена задача разработки алгоритма поиска и выделения объектов на основе информации о цвете, обладающего следующими особенностями:

- Устойчивость по отношению к изменению условий освещения (хотя бы частичное решение первой проблемы).
- Цвет объекта задается человеком.

Алгоритм состоит из следующих этапов:

- Получение изображения с видеокамеры.
- Перевод изображения из цветового пространства RGB в HSV.
- Фильтрация по цвету (настройка цветовой маски).

- Проведение основных морфологических преобразований – растягивание и сужение.
- Нахождение контуров найденного объекта.
- Отрисовка метки.

К преимуществам данного алгоритма можно отнести следующие особенности:

- Человек выбирает только диапазон по шкале H, и в результате сразу формируется трехмерная область в цветовом пространстве;
- Человек не ограничен набором тонов, соответствующих цвету или цвету, производному из основных, он может выбирать любой диапазон цветовых тонов на шкале H, даже объединяя цветочные тона со смежными базовыми цветами в одном диапазоне;
- Ввод информации о цвете выполняется не словесным образом (текстом или устно), а выбором диапазона цветовых тонов на визуально отображаемой шкале цветовых тонов, и это позволяет избежать индивидуальных различий в понимании цветовых терминов естественного языка.

Основным недостатком данного способа можно считать привязку только к сравнительно ярким и насыщенным оттенкам, что определяется ограничением на основе значений $V_{\min}$, $S_{min}$. Следствиями этого являются:

- Невозможность выделения цветовой области для темного цвета, например, для такого базового цвета, как коричневый;
- Невозможность выделения цветовой области для цветов, относящихся к градациям серого, в том числе для черного и белого цветов.

Для реализации алгоритма было решено написать программу с использованием библиотеки OpenCV.

Для получения изображения с камеры OpenCV предоставляет простой способ работы с веб-камерами (модуль HighGui) (см. листинг 4.1)

Листинг 4.1 – Получение изображения с камеры.

```
#include <opencv2/highgui.hpp>

int main {
    capture.ope
    n(0);
    capture.set(CV_CAP_PROP_FRAME_WIDTH,
    FRAME_WIDTH);
    capture.set(CV_CAP_PROP_FRAME_HEIGHT,
    FRAME_HEIGHT);
}
```

Перевод изображения в цветовую модель HSV обусловлен тем, что эта модель очень удобна для поиска на изображении объектов по цвету.

Среди преимуществ модели HSV можно выделить следующие:

- Отделение цветового компонента в виде цветового тона *(H)*, при этом значение тона не зависит от изменений интенсивности внешнего освещения, если оно является белым;
- Удобство для решения проблемы именования цветов с помощью естественного языка, так как эта проблема очень часто рассматривается на основе моделей, где выделяются компоненты цветового тона и насыщенности цвета, и даже если применяются другие модели, то все равно обычно выполняется сопоставление с базовой моделью Манселла, которая в указанном смысле похожа на модель HSV.

Учитывая все это, в качестве основной была выбрана модель HSV (перевод см. в листинге 4.2).

Листинг 4.2 – Перевод из RGB в HSV и преобразование в черно-белое изображение.

```
cvtColor(cameraFeed, HSV, COLOR_BGR2HSV);
inRange(HSV, Scalar(H_MIN, S_MIN, V_MIN),
Scalar(H_MAX, S_MAX, V_MAX), threshold);
```

Следующий шаг - проведение основных морфологических преобразований. Это требуется для избавления от случайных вкраплений на

изображении, шума и объединения областей, разделенных тенью (см. листинг 4.3).

Листинг 4.3 – Морфологические преобразования

```
void morphsOps(Mat &thresh) {
Mat erodeElement = getStructuringElement(MORPH_RECT, Size(3, 3));
Mat dilateElement = getStructuringElement(MORPH_RECT, Size(8, 8));

erode(thresh, thresh, erodeElement);
erode(thresh, thresh, erodeElement);

dilate(thresh, thresh, dilateElement);
dilate(thresh, thresh, dilateElement);

}
```

Нахождение контуров приведено в листинге 4.4.

Листинг 4.4 – Нахождение и сравнение контуров.

```
Mat temp; threshold.copyTo(temp);


vector< vector<Point> > contours; vector<Vec4i> hierarchy;


findContours(temp, contours, hierarchy, CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE);



double refArea = bool objectFound

0;
= false;



if (hierarchy.size() > 0) {
int numObjects = hierarchy.size(); if (numObjects < MAX_NUM_OBJECTS) {
for (int index = 0; index >=0; index = hierarchy[index][0]) { Moments moment =
```

```
moments((cv::Mat)contours[index]); double area =
moment.m00;


if (area > MIN_OBJECT_AREA && area <
MAX_OBJECT_AREA && area > refArea) {
x = moment.m10/area; y = moment.m01/area;
objectFound = true; refArea = area;
} else
objectFound = false;}
if (objectFound = true) {
putText(cameraFeed, "Tracking object", Point(0,
50), 2, 1, Scalar(0, 255, 0), 2);
drawObject(x, y, cameraFeed); }
} else
putText(cameraFeed, "Too much noise! Adjust
filter", Point(0, 50), 1, 2, Scalar(0, 255, 0),
2);
}
```

### 4.1.2 Тестирование

Тестирование реализованного алгоритма проходило на ноутбуке msi PE60 6QE с процессором Intel Core i7 и со встроенной веб-камерой, разрешение которой составляет 640x480 пикселей. На вход видеопотока подавались различные объекты разных цветов. Как уже говорилось, для каждого объекта требуется специально настраивать цветовую маску. Для настройки этой маски, а именно параметров `h_min`, `h_max` функции `inRange` пришлось сделать отдельное окно с множеством бегунков. Чтобы выделить нужный цвет, необходимо подобрать границы компонента *H* (Hue). Параметр *S* (Saturation) отвечает за насыщенность цвета. *V* (Value) определяет яркость цвета. Затененный объект будет иметь низкое значение V.

При запуске программы появляется 4 окна.

Первое окно – оригинальное изображение с веб-камеры, без каких-либо преобразований. Именно на нем будет помечаться распознанный объект (Рисунок 4.1)

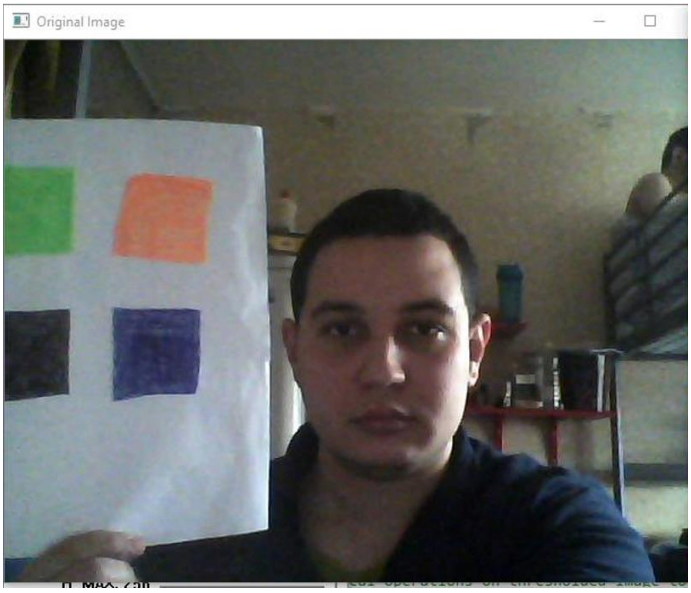


Рисунок 4.1 - Оригинальное изображение

Второе окно – изображение с веб-камеры в цветовом пространстве HSV. Оно нужно для демонстрации того, что данное цветовое пространство действительно легче воспринимается глазом для выделения конкретного объекта, следовательно, в нем проще создать правильную маску для выделения нужного цвета (рисунок 4.2)

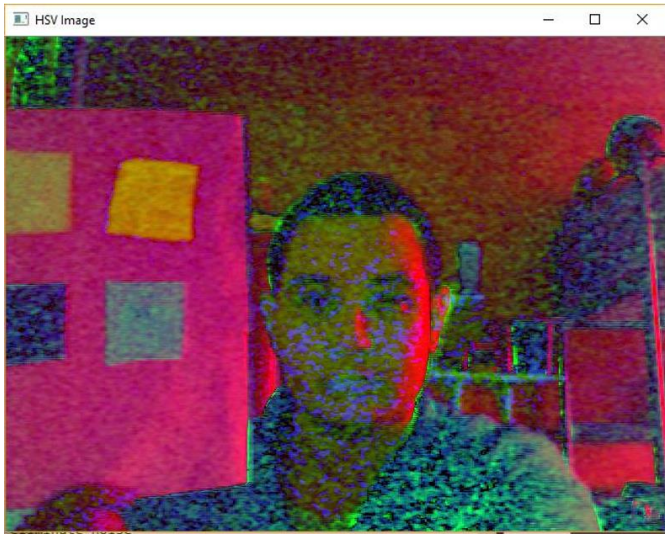


Рисунок 4.2 – HSV изображение

Третье окно – изображение с веб-камеры после преобразования цветной картинки в черно-белую маску. В этой маске, все пиксели, попадающие в заданный диапазон - становятся белыми. Прочие - черными. (рисунок 4.3)


Рисунок 4.3 - Демонстрация окна с черно-белым изображением

И наконец, четвертое окно – ползунки, с помощью которых мы будем настраивать цветовую маску (рисунок 4.4)

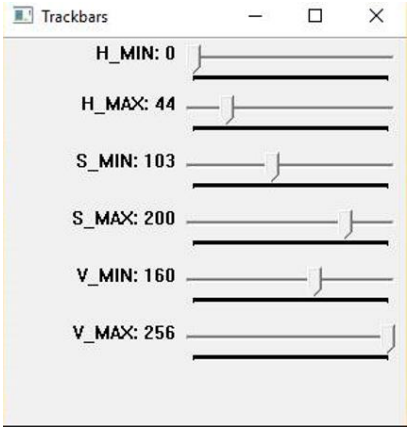


Рисунок 4.4 - Демонстрация окна с ползунками

Запускаем код программы и проверяем результат. Программа запушена успешно и начинает преобразовать(фильтровать) изображения так чтобы она распознавала оранжевый квадрат. (рисунок 4.5)

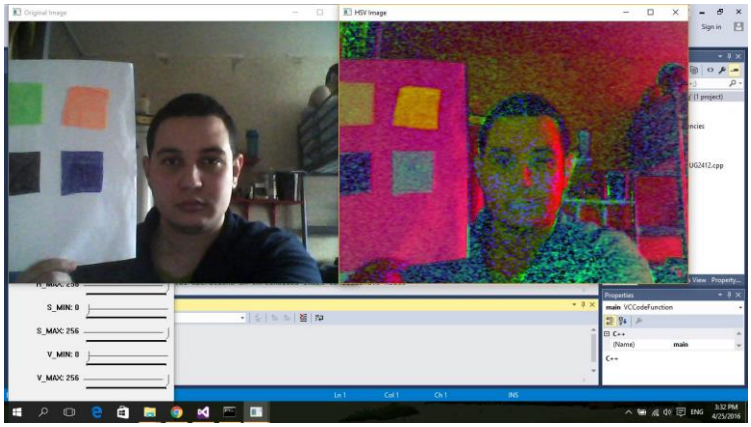

Рисунок 4.5 – HSV

После преобразования оранжевого квадрата мы получаем результат, который показывает, что алгоритм позволяет корректно определить объект в пространстве. (Рисунок 4.6)

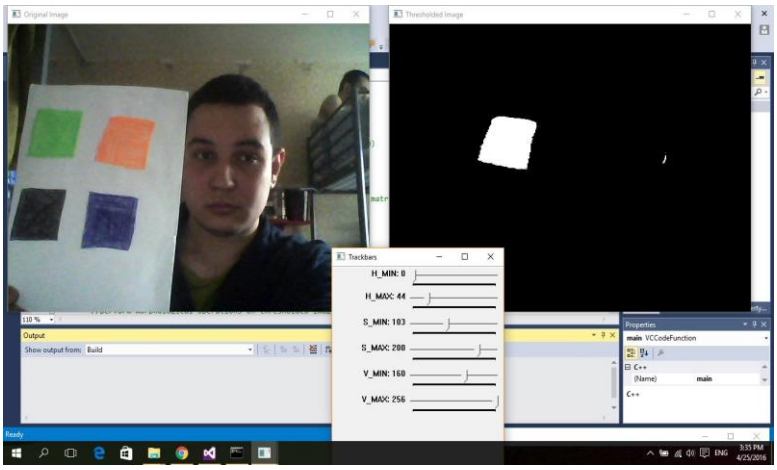


Рисунок 4.6 - Фильтрование цвета

Результат показывает, что алгоритм позволяет корректно определить объект в пространстве.

## 4.2 Отслеживание цветного объекта

### 4.2.1 Реализация алгоритма

В этом разделе мы будем отслеживать объект после его преобразования (фильтраций) на конкретном цветовом диапазоне. Для этого будем использовать track bar (рисунок 4.4), который был реализован в предыдущем разделе.

Создадим функцию (метод) который будет отслеживать объект после его преобразования.

Листинг 4.5 – Отслеживание фильтрованного объекта.

```
void trackFilteredObject(int &x, int &y, Mat threshold, Mat
&cameraFeed) {

      Mat temp;
      threshold.copyTo(temp);
      //these two vectors needed for output of findContours
```

```
      vector< vector<Point> > contours;
      vector<Vec4i> hierarchy;
      //find contours of filtered image using openCV findContours
function

      findContours(temp,    contours,    hierarchy,    CV_RETR_CCOMP,
CV_CHAIN_APPROX_SIMPLE);
      //use moments method to find our filtered object
      double refArea = 0;
      bool objectFound = false;
      if (hierarchy.size() > 0) {
      int numObjects = hierarchy.size();
      //if number of objects greater than MAX_NUM_OBJECTS we have a
noisy filter
      if (numObjects<MAX_NUM_OBJECTS) {
            for (int index = 0; index >= 0; index = hierarchy[index][0])
{

      Moments moment = moments((cv::Mat)contours[index]);
                        double area = moment.m00;

      if (area>MIN_OBJECT_AREA && area<MAX_OBJECT_AREA && area>refArea)
{
                              x = moment.m10 / area;
                              y = moment.m01 / area;
                              objectFound = true;
                              refArea = area;
                        }
                        else objectFound = false;
                  }
                  //let user know you found an object
                  if (objectFound == true) {
                        putText(cameraFeed, "Tracking Object", Point(0,
50), 2, 1, Scalar(0, 255, 0), 2);
                        //draw object location on screen
                        drawObject(x, y, cameraFeed);
                  }
            }
            else putText(cameraFeed, "TOO MUCH NOISE! ADJUST FILTER",
Point(0, 50), 1, 2, Scalar(0, 0, 255), 2);
      }
```

### 4.2.2 Тестирование

Запускаем код программы и проверяем результат. Программа запушена успешно и начинает фильтровать несколько разных объектов (100 рублей, iPhone, кольцо, коробка). Результаты демонстрируются на рисунках 4.7, 4.8, 4.9, 4.10.


Рисунок 4.7 – Отслеживание 100 рублей

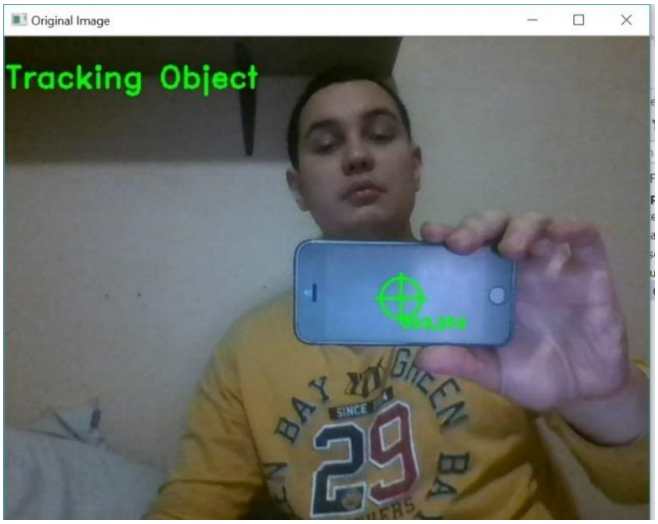


Рисунок 4.8– Отслеживание телефон.

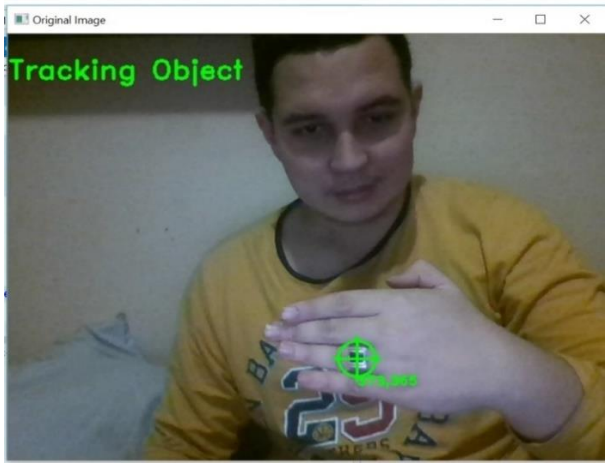


Рисунок 4.9 – Отслеживание кольца.

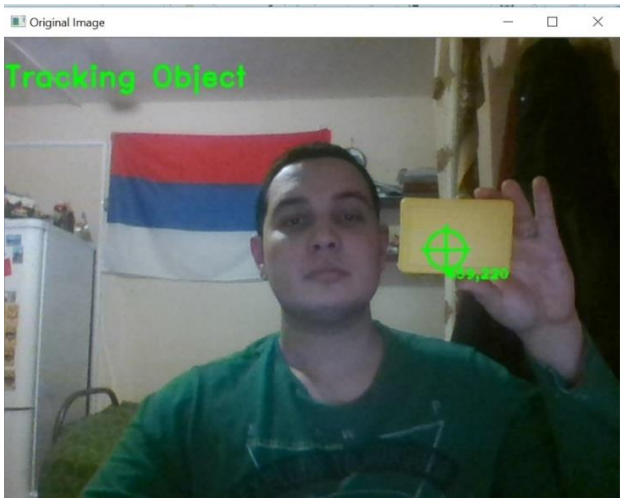


Рисунок 4.10 – Отслеживание коробки

При изменении положения объекта в пространстве метка корректно следует за ним.

Далее мы будем проводить тестирование при двух условиях освещения: хорошее освещение (дневной свет + источник света) и плохое освещение (ночь, без источника света).

В результате проведённого эксперимента были получены значения для каждой компоненты. Они приведены в таблицах 4.1 и 4.2.

Таблица 4.1 – Значения компонентов HSV при хорошем освещении.

| | Hue(min, max) | Saturation(min, max) | Value(min, max) |
|---|---|---|---|
| Зеленый объект | (26, 41) | (111, 256) | (0, 256) |
| Желтый объект | (13, 144) | (141, 256) | (39, 235) |
| Красный объект | (0, 66) | (179, 256) | (108, 149) |

Таблица 4.2 – Значения компонентов HSV при плохом освещении.

| | Hue(min, max) | Saturation(min, max) | Value(min, max) |
|---|---|---|---|
| Зеленый объект | (49, 57) | (43, 256) | (39, 173) |
| Желтый объект | (29, 44) | (96, 209) | (104, 256) |
| Красный объект | (62, 256) | (37, 96) | (0, 256) |

При правильном выборе цветовой модели алгоритм устойчив по отношению к изменению условий освещения.

Следующим шагом рассмотрим вероятность детектирование объекта. Мы будем отмечать количество прохождений объекта через каждый кадр.

Программа должна отслеживать объект и при нахождении рисовать круг на найденном объекте. Далее мы смотрим сколько раз при каждом кадре рисуется круг, то есть сколько раз программа будет находить объект при каждом кадре. Добавляем два счетчика в коде.

Листинг 4.2 Счетчик для нахождения объекта.

```
if (numObjects<MAX_NUM_OBJECTS){
      int counter1 = 0;
      for (int index = 0; index >= 0; index = hierarchy[index][0]) {
      Moments moment = moments((cv::Mat)contours[index]);
      double area = moment.m00;
      if(area>MIN_OBJECT_AREA && area<MAX_OBJECT_AREA &&
area>refArea){
      x = moment.m10 / area;
      y = moment.m01 / area;
      objectFound = true;
      counter1++;
      refArea = area;
                        }
```

Листинг 4.3 Счетчик рисования метки для отслеживания объекта.

```
if (objectFound == true){
      putText(cameraFeed, "Tracking Object", Point(0, 50), 2,
1,Scalar(0, 255, 0), 2);
                        drawObject(x, y, cameraFeed);
                        counter2++;
                  }
```

Запускаем код программы и проверяем результат. Программа запушена успешно. Из рисунка 4.11 мы видим два столбца.

Первый столбец - это found object (нахождение объекта), который будет равняться 1 при детектировании объекта.

Второй столбец - это draw object (рисовать объект), который будет равняться 1 при рисовании круга вокруг найденного объекта во время отслеживания.

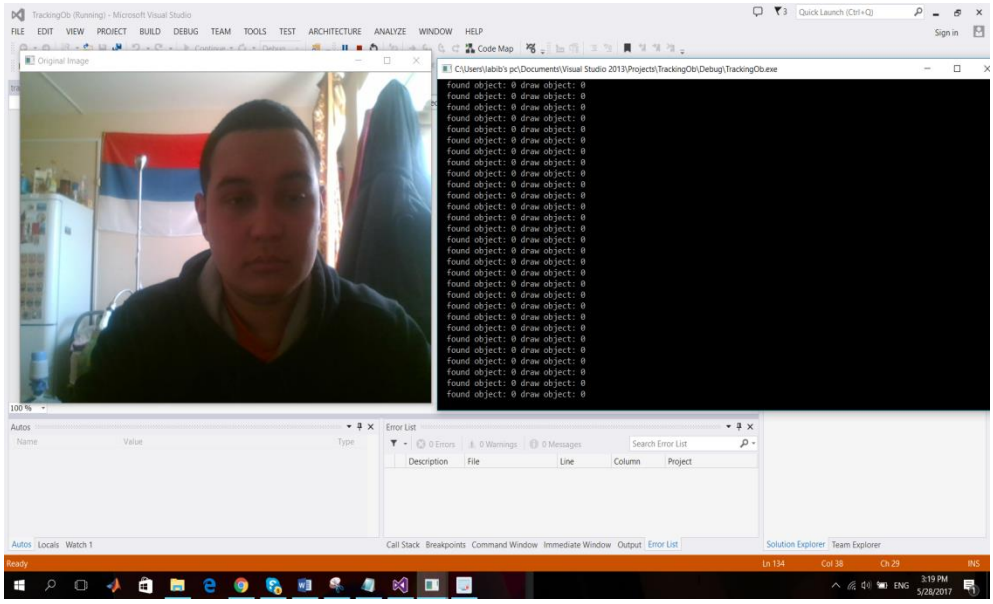

Рисунок 4.11 – Found object и draw object

Из рисунка 4.12 мы видим, что на нахождение объекта влияет освещение. Программа находить объект, но не рисует круг нахождения объекта при каждом детектировании. Причиной является плохое освещение.

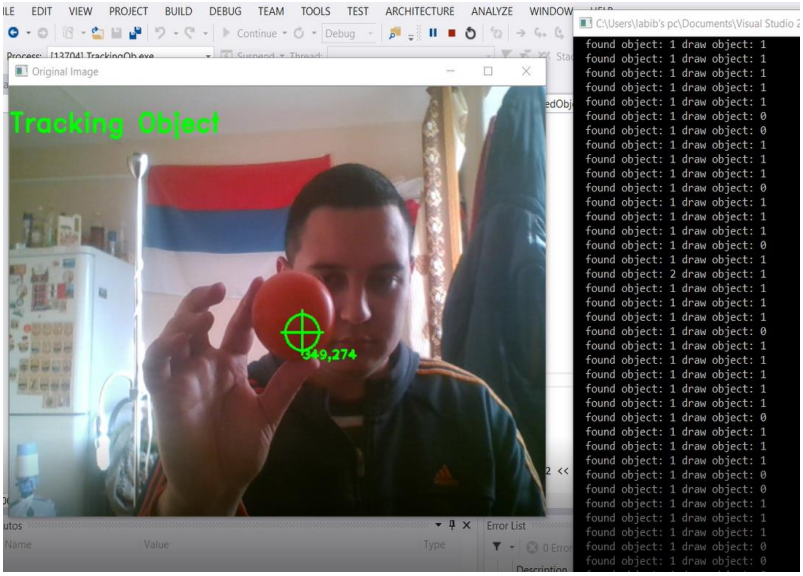


Рисунок 4.12 – Детектирование объекта при плохом освещении

Проведем тот же самый эксперимент, но при хорошем освещении (рисунок 4.13).

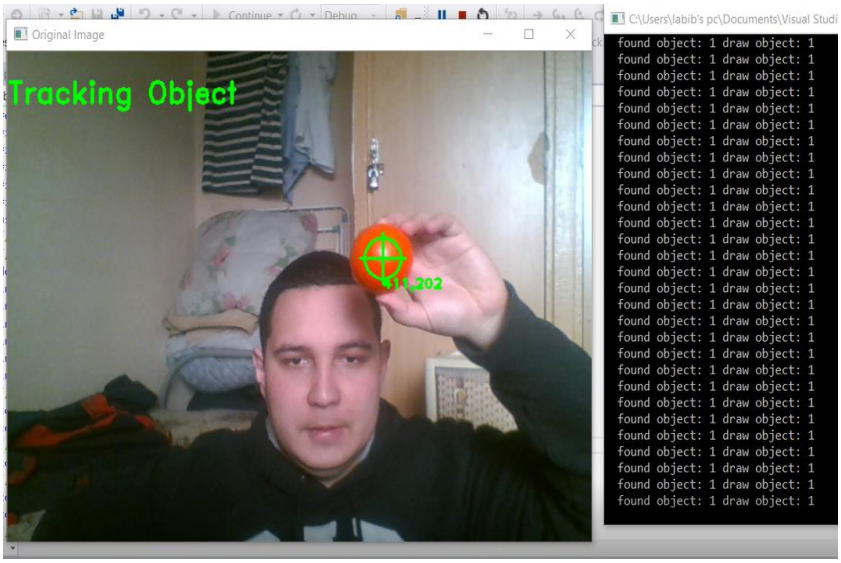


Рисунок 4.13 - Детектирование объекта при хорошем освещении

Результат показывает, что объект при хорошем освещении детектируется с 100% точностью.

Теперь посчитаем вероятность детектирования при плохом освещении.

Вероятность детектирования при плохом освещении:

$$\frac{draw\ object}{Found\ object} = \frac{25}{42} = 0{,}609 = 60{,}9\%.$$

## 4.3 Отслеживание несколько объектов

Этот раздел посвящен отслеживанию несколько объектов. Объектами отслеживания будут два оранжевых шара. Задачей нашей программы будет нахождение этих объектов, вычисление их координат и радиусов шаров.

### 4.3.1 Реализация алгоритма

Для того чтобы программа отслеживала шары, мы должны реализовать код, который будет находить центр этих шаров и вычислять положение центра шара и показывать длину радиуса шара.

Листинг 4.6 – Центр и радиус шара.

```
for (int i = 0; i < v3fCircles.size(); i++) {// for each
circle.                        // show ball position x, y, and
radius to command line

     std::cout << "ball position x = " << v3fCircles[i][0]


       // x position of center point of circle

     << ", y = " << v3fCircles[i][1]

      //y position of center point of circle

     << ", radius = " << v3fCircles[i][2] << "\n";

       // radius of circle

      // draw small green circle at center of detected object

      cv::circle(imgOriginal,
               // draw on original image

     cv::Point((int)v3fCircles[i][0], (int)v3fCircles[i][1]),

       // center point of circle

      3,
          // radius of circle in pixels

     cv::Scalar(0, 255, 0),
               // draw pure green (remember, its BGR, not RGB)

     CV_FILLED);
               // thickness, fill in the circle

       // draw red circle around the detected object

     cv::circle(imgOriginal,
               // draw on original image

     cv::Point((int)v3fCircles[i][0], (int)v3fCircles[i][1]),

       // center point of circle

     (int)v3fCircles[i][2],
               // radius of circle in pixels

     cv::Scalar(0, 0, 255),
               // draw pure red

     3);
          // thickness of circle in pixels
```

```
    }     // end for
```

### 4.3.2 Тестирование

Запускаем код программы и проверяем. Программа запушено успешно и как мы видим на рисунке 4.14 т рисунок 4.15 что программа выделает объект(шары) и показывает их координаты и еще радиус шара.

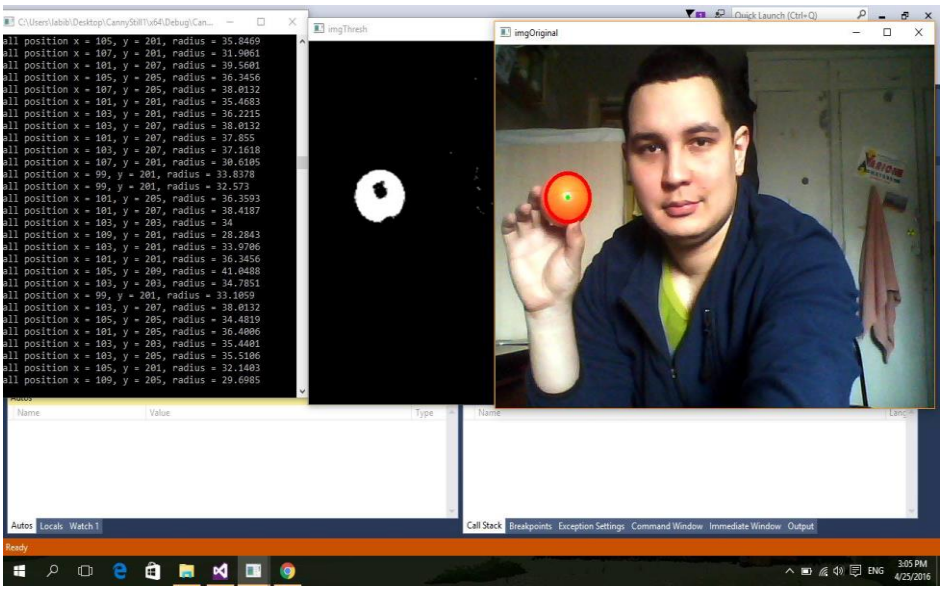


Рисунок 4.14 – Отслеживание одного объекта

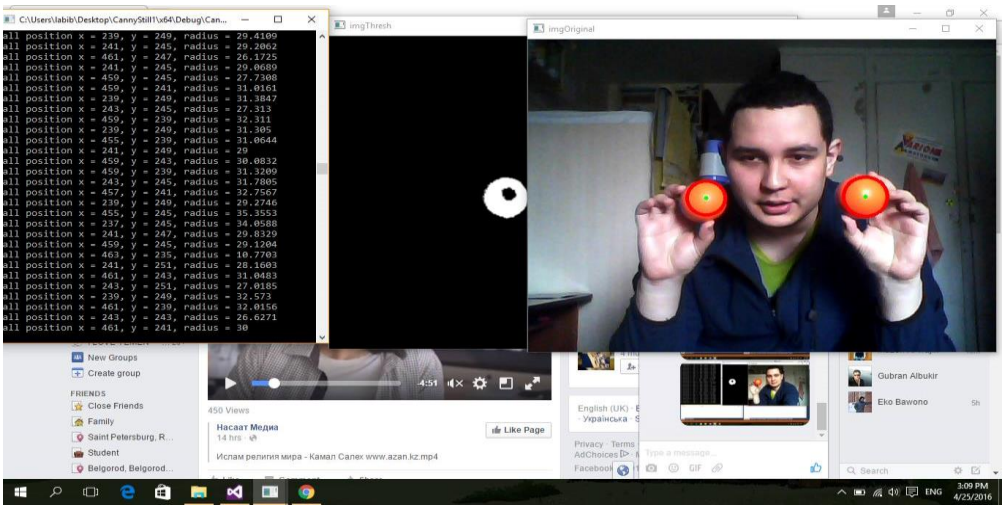


Рисунок 4.15 – Отслеживание несколько объектов

Результат показывает при изменении положения объекта в пространстве метка корректно следует за ним.

# 5 Анализ изображений

## 5.1 Теоретическая часть

### 5.1.1 Определения и формулы математического маятника

Математический маятник – это материальная точка, подвешенная на длинной невесомой нерастяжимой нити. (Рисунок 5.1)

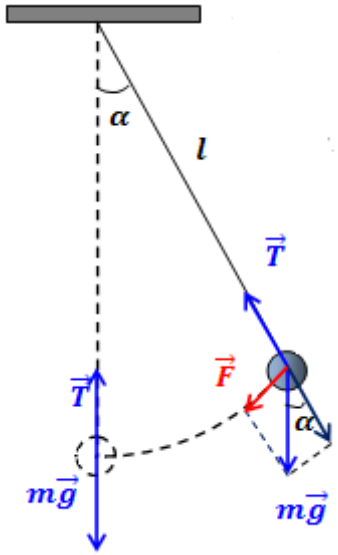


Рисунок 5.1 - Математический маятник.

Математический маятник – это модель системы, совершающей гармонические колебания. Свободные колебания математического маятника при малых углах отклонения описываются уравнением гармонических колебаний.

В положении равновесия сила тяжести и сила упругости нити уравновешивают друг друга, и материальная точка находится в покое. При отклонении материальной точки от положения равновесия на малый угол $\alpha$ на тело будет действовать возвращающая сила $\vec{F}$ , которая является тангенциальной составляющей силы тяжести:

$$F = mg * \sin(\alpha) \quad (5.1)$$

Эта сила сообщает материальной точке тангенциальное ускорение, направленное по касательной к траектории, и материальная точка начинает двигаться к положению равновесия с возрастающей скоростью. По мере приближения к положению равновесия возвращающая сила, а, следовательно, и тангенциальное ускорение точки, уменьшаются. В момент прохождения положения равновесия угол отклонения $\alpha = 0$, тангенциальное ускорение также равно нулю, а скорость материальной точки максимальна. Далее материальная точка проходит по инерции положение равновесия и, двигаясь в направлении, противоположном силе $\vec{F}$, сбавляет скорость. В крайнем положении материальная точка останавливается, и затем начинает двигаться в обратном направлении.

Период колебаний математического маятника:

$$T = 2\pi\sqrt{\frac{l}{g}} \quad (5.2)$$

Период колебаний математического маятника не зависит от массы груза и амплитуды колебаний.

### 5.1.2 Основные формулы математического маятника

Для решения поставленной задачи необходимо определить основные формулы математического маятника, которые нам понадобиться для решения нашей поставленной задачи. (рисунок 5.2)

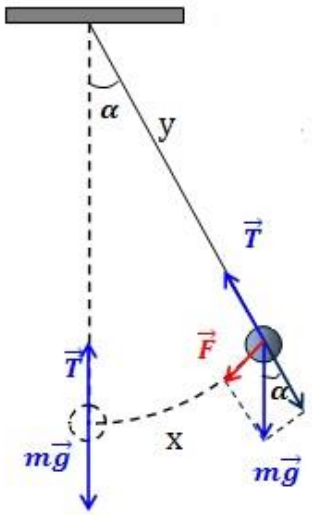


Рисунок 5.2 - Математический маятник поставленной задачи.

Из рисунка 5.2 мы можем вывести следующие формулы:

$$\begin{cases} \vec{T} = mg * \cos(\alpha) \\ \vec{F} = -mg * \sin(\alpha) = m * a \end{cases} \quad (5.3)$$

Где $\vec{T}$ – это сила натяжения нити. Из формулы (5.3) мы можем вывести ускорения из уравнения силы $\vec{F}$ где мы делим уравнения на массу и получим следующая:

$$a = -g * \sin(\alpha) \quad (5.4)$$

Вычисляем угол sin $(\alpha)$ и получаем формулу:

$$\alpha = \sin(\alpha) = \frac{x}{y} \quad (5.5)$$

Где $x$- это расстояние который проходить маятник, а $y$- это длина нити.

Поставляем формулу (5.5) в (5.4) и получаем следующею формулу для расчета ускорение:

$$a = -g * \frac{x}{y} \quad (5.6)$$

Сейчас будем преобразовать формулу так чтобы мы нашли угловую скорость $\omega$ для нашей поставленной задачи из уравнения гармонической колебания:

$$a = -\omega^2 * x \tag{5.7}$$

Преобразуем уравнение (5.7) так чтобы она подходила к нашей поставленной задачи:

$$\omega^2 = \frac{g}{y} \Rightarrow \omega = \sqrt{\frac{g}{y}} \tag{5.8}$$

Из уравнения (5.8) мы можем найти период движения маятника $T$:

$$\omega = \sqrt{\frac{g}{y}} \Rightarrow 2\pi f = \sqrt{\frac{g}{y}} \tag{5.9}$$

Где $f$- это частота маятника и получаем следившею формулу:

$$f = \frac{1}{2\pi}\sqrt{\frac{g}{y}} \tag{5.10}$$

Как мы знаем, что период колебаний $T$ — это наименьший промежуток времени, за который маятник совершает одно полное колебание (то есть возвращается в то же состояние, в котором он находился в первоначальный момент, выбранный произвольно), и при этом у нас будет следующие формула для периода:

$$T = \frac{1}{f} \tag{5.11}$$

Поставляем уравнения (5.10) в уравнение (5.11) и получаем формулу периода для нашей поставленной задачи (Формула Гюйгенса):

$$T = 2\pi\sqrt{\frac{y}{g}} \tag{5.12}$$

## 5.2 Практическая часть

В этом разделе будет реализована программа математического маятника с использованием библиотеки Open CV 2.4.9 и языка программирования C++.

### 5.2.1 Реализация программы математического маятника

Задачей раздела будет реализация программы, которая будет исследовать движения маятника с помощью компьютерного зрения. Для исследования движений маятника будут применены метод фильтрации цвета и метод нахождения контуров. Код программы будет написан на языке программирования C++ с использованием библиотеки OpenCV 2.4.9 и средой разработки Microsoft Visual Studio 2013 Ultimate.

Первым шагом будет реализация алгоритма для генерации HSV матрицы.

Листинг 5.1 Генераций HSV матрицу.

```
// Generates HSV Matrix
cv::cvtColor(image_frame,   // Input image
image_HSV, // output image in HSV
CV_BGR2HSV); // constant refering to color space
transformation
```

Следующим шагом будет реализация алгоритма для фильтрации изображений по цвету.

Листинг 5.2 Фильтрация изображений по цвету.

```
//filtering image for colors
cv::inRange(image_HSV, //input image to be filtered
cv::Scalar(0, 155, 155), //min thresshold value
cv::Scalar(18, 255, 255), // max threshold value
image_Color1); //output image
```

```
cv::inRange(image_HSV, //input image to be filtered
cv::Scalar(144, 189, 0), //min thresshold value
cv::Scalar(177, 255, 255), // max threshold value
image_Color2); //output image
```

Затем нужно будет написать алгоритм, который будет находить контуры для исследуемого объекта.

Листинг 5.3 Нахождение контуров.

```
/// Find contours
findContours(image_Color1, // input image
contours_1, // vector to save contours
hierarchy_1,
CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE,
cv::Point(0, 0));

findContours(image_Color2, // input image
contours_2, // vector to save contours
hierarchy_2,
CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE,
cv::Point(0, 0));
```

Запускаем код программы и проверяем результат. Как видно из рисунков 5.3 и 5.4 программа запушена успешно и отлеживает движения математического маятника, показывает нам центр шара, ось маятника и вычисляет координаты движения маятника.

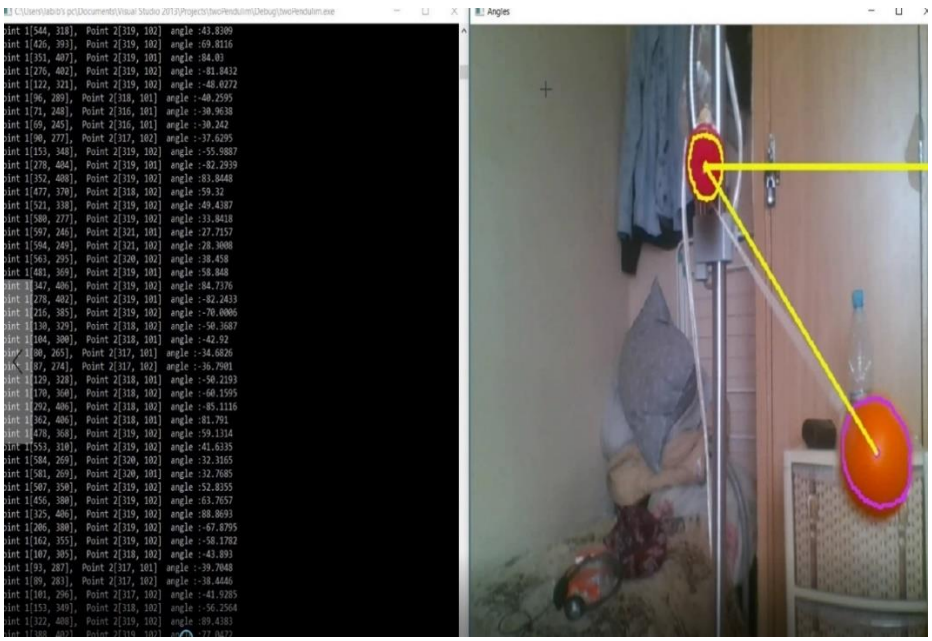


Рисунок 5.3 - Движение математического маятника.

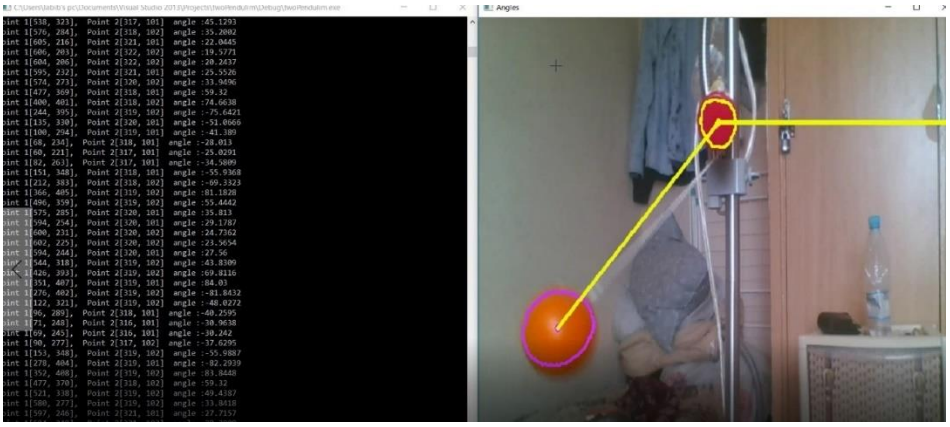


Рисунок 5.4 -Движение математического маятника 2.

## 5.3 Анализ соответствия реализованного решения поставленной задачи

В этом разделе мы будем проводить анализ движения маятника, используя законы физики. Будет проведен сравнительный анализ маятника и вычислены некоторые его компоненты. На основе полученных результатов будет проведен анализ, который поможет вычислить геометрические параметры исследуемой системы. Затем будет построена эквивалентная

математическая модель с физической моделью и сравнения их с соответствующий моделью.

### 5.3.1 Вычисление математического маятника

Первое что мы будем делать это вычислим длину нити для нашего маятника через следующую формулу:

$$y = \sqrt{(x_b^* - x_a^*)^2 + (y_b^* - y_a^*)^2} \tag{5.13}$$

Где $x_b^*$, $x_{a,}^*$ $y_b^*$, $y_a^*$- координатные точки.

$$x_b^* = 318, x_a^* = 340, y_b^* = 100, y_a^* = 407.$$

Поставляем значение в формулу (5.13) и получаем что у нас $y$ =307,78 пиксел и это будить приблизительно 8 сантиметров или 0,08 метров.

Далее вычисляем расстояние, которое проходить у нас маятник. Преобразовываем формулу (5.13) и получаем следующую формулу:

$$x = \sin(\alpha) * y \tag{5.14}$$

Даны четыре угла $90^0, 60^0, 45^0, 30^0$. Вычисляем $x$ при четырёх углах и получаем следующие значение:

$$x_1 = \sin(90^0) * 0{,}08 = 0{,}08\ m$$
$$x_2 = \sin(60^0) * 0{,}08 = 0{,}069\ m$$
$$x_3 = \sin(45^0) * 0{,}08 = 0{,}056\ m$$
$$x_4 = \sin(30^0) * 0{,}08 = 0{,}04\ m$$

Затем вычисляем ускорения, частоты колебания и периода, когда маятник приходит в движение используя формулы (5.6), (5.10), (5.12).

Подсчитаем значение ускорения:

$$a_1 = -9{,}8 * \frac{0.08}{0.08} = -9{,}8\ m/c^2$$

$$a_2 = -9{,}8 * \frac{0.069}{0.08} = -8{,}45\ m/c^2$$

$$a_3 = -9{,}8 * \frac{0.056}{0.08} = -6{,}86\ m/c^2$$

$$a_3 = -9{,}8 * \frac{0.04}{0.08} = -4{,}8\ m/c^2$$

Подсчитаем значение частоты:

$$f = \frac{1}{2 * \pi} * \sqrt{\frac{9{,}8}{0{,}08}} = 1{,}76 \text{ герц}$$

Подсчитаем значение периода:

$$T = 2 * \pi * \sqrt{\frac{0{,}08}{9{,}8}} = 0{,}56 \text{ с}$$

### 5.3.2 Сравнительный анализ математической модель с физической модель маятника

В этом разделе будет проведен сравнительный анализ экспериментальных и расчетных значений колебаний математического маятника. Для расчетных значений будем использовать программу Matlab. Затем полученные расчетные значения будем сравнивать с экспериментальными значениями. Вычислим погрешность и посмотрим насколько надежен выбранный метод и алгоритм для симуляций математического маятника.

Первый шаг – это реализация кода на Matlab-е. Запускаем код на Matlab-е и как мы видим из рисунков 5.5 и 5.6 программа запустилось успешно и работает хорошо.

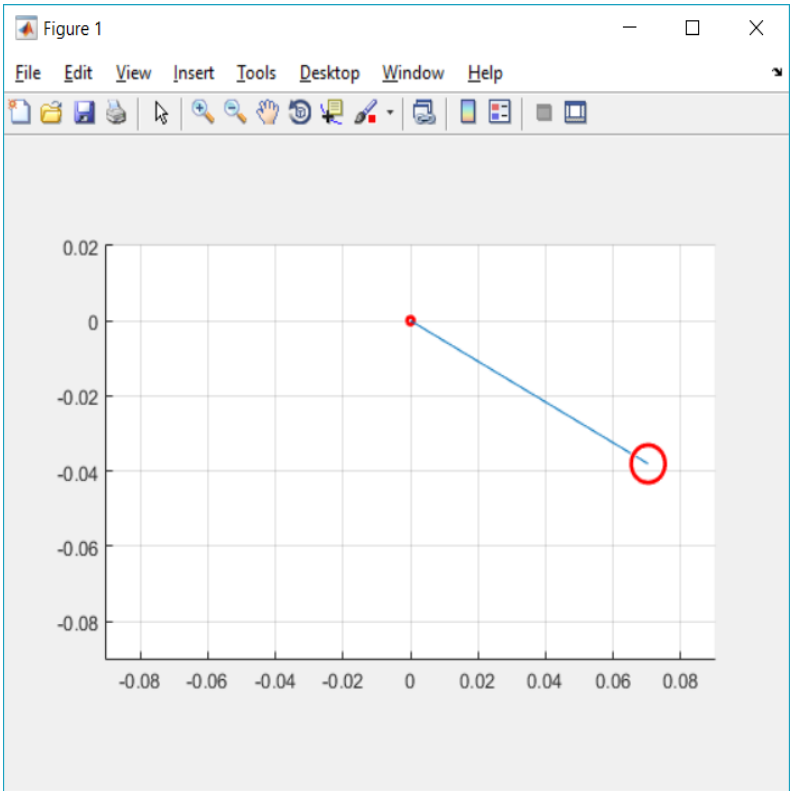


Рисунок 5.5 - Симуляция математического маятника на Матлаб-е.

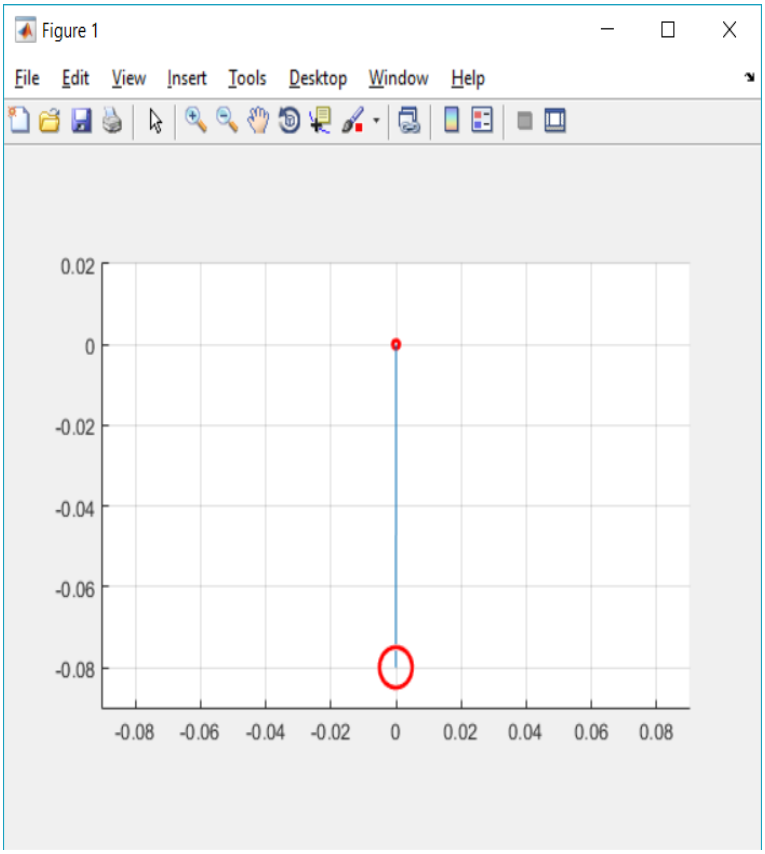


Рисунок 5.6 – Маятник остановился.

Следующий шаг - подсчитываем на Matlab-е и на C++ число полных колебаний маятника до достижения им состояния равновесия (остановки).

Число полных колебания расчётного маятника = 130.

Число полных колебания экспериментального маятника = 122.

Теперь вычисляем погрешность по следующей формуле:

$$\Delta = \frac{n_{\text{расч}} - n_{\text{экспе}}}{n_{\text{расч}}} * 100\% \quad (5.15)$$

Подставляем значения в формулу (5.15) и получаем следующую погрешность:

$$\Delta = \frac{130 - 122}{130} * 100\% = 6{,}15\%$$

Результат анализа показывает, что программа симуляций маятника на C++ хорошая, так как погрешность не превосходить 10%.

Погрешность равна 6,15%. Этот результат говорит о том, что человеческий фактор действует на систему (например, вычисление расчетных формул) и что наша система находиться не в идеальных условиях.

# Выводы

В время выполнение выпускной работы были получены следующие результаты:

- Алгоритм отслеживание объекта позволяет корректно определить объект в пространстве.
- При изменении положения объекта в пространстве метка корректно следует за ним;
- Благодаря правильному выбору цветовой модели алгоритм устойчив по отношению к изменению условий освещения.
- Результат анализа показывает, что программа симуляций маятника на C++ хорошая так как погрешность не превосходить 10%.
- Погрешность математического маятника равно 6,15% говорить о том, что наша система действует на неё человеческие факторы (например, вычисление по расчетные формулы) а также наша система находиться в не идеальных условиях.

# Заключение

В рамках выполнения выпускной работы был исследован алгоритм распознавания объектов по цвету и был пройден ряд этапов, в ходе которых были достигнуты следующие результаты:

- Проведен анализ предметной области: рассмотрены основные понятия, определения и дальнейшие перспективы развития компьютерного зрения.
- Был выбран инструментарий, изучены особенности его применения и использование определенных навыков и знаний, полученных в ходе обучения в университете.
- В ходе исследования алгоритма была выполнена его реализация и тестирование.
- Проведено исследование математического маятника и на основе полученных результатов проведен анализ для вычисление физических значений.
- Были построены эквивалентные математическая модель маятника и физическая модель.
- Проведён сравнительный анализ с соответствующей моделью.

# Список литературы


1. Форсайт Д., Понс Ж. Компьютерное зрение - современный подход. М.: «Вильямс», 2004- 928 с.
2. Шапиро Л.,Стокман Д. Компьютерное зрение.М.:БИНОМ,2006 – 752с.
3. Wiki – Техническое зрение. [Электронный ресурс] URL: http://wiki.technicalvision.ru/ (дата обращения: 26.09.2016)
4. Computer Vision System Toolbox. Разработка и моделирование систем компьютерного зрения и обработки видео. [Электронный ресурс] URL: http://matlab.ru/products/computer-vision-system-toolbox/computer-vision-system-toolbox_rus.pdf (дата обращения: 05.03.2017)
5. MATLAB [Электронный ресурс] URL: http://matlab.ru/products/matlab (дата обращения: 02.03.2017)
6. Robocraft. OpenCV шаг за шагом. Введение. [Электронный ресурс] URL: http://robocraft.ru/blog/computervision/264.html (дата обращения: 11.09.2016)

   OpenCV шаг за шагом. Обработка изображения - детектор границ Кенни (Canny). [Электронный ресурс] URL: http://robocraft.ru/blog/ computervision/484.html (дата обращения: 26.09.2016)
7. OpenCV шаг за шагом. Нахождение контуров и операции с ними. [Электронный ресурс] URL: http://robocraft.ru/blog/ computervision/ 640.html (дата обращения: 16.10.2016)
8. Детектирование объектов - поиск объекта по шаблону (Template matching) [Электронный ресурс] URL: http://robocraft.ru/blog /computervision /3046.html (дата обращения: 10.11.2016)
9. Gary Bradski, Adrian Kaehler . Learning OpenCV // O'Reilly Media, 2008, P.580
10. Batavia P., Singh S. Obstacle detection using adaptive color segmentation and color stereo homography // IEEE International Conference on Robotics and Automation. - 2001. - Vol. 1. - P. 705-710.

11. Tsai L., Hsieh J., Fan K. Vehicle detection using normalized color and edge map // 18th IPPR Conference on Computer Vision, Graphics and Image Processing. -Taipei, 2005. - P. 849-856.
12. Fleyeh H. Road and traffic sign color detection and segmentation - A Fuzzy Approach // IAPR Conference on Machine Vision Applications. - Tsukuba Science City, 2005. - P. 124-127.
13. Fuzzy-based algorithm for color recognition of license plates / F. Wang, L. Man, B. Wang, Y. Xiao, W. Pan, X. Lu // Pattern Recognition Letters. - 2008. - Vol. 29. - P. 1007-1020.
14. Sural S., Qian G., Pramanik S. Segmentation and histogram generation using the HSV color space for image retrieval // Proceedings 2002 International Conference on Image Processing. - New York, 2002. - Vol. 2. - P. 589-592.
15. Mojsilovic A. A computational model for color naming and describing color composition of images // IEEE Transactions on Image Processing. - 2005. - Vol. 14. - P. 690-699.
16. Csink L., Paulus D., Ahlrichs U., Heigl B. Color normalization and object localization // Vierter Workshop Farbbildverarbeitung. - Koblenz. - 1998. - P. 49.

# Приложение А

Отслеживание объекта через фильтрации света

```
#define _CRT_SECURE_NO_DEPRECATE

#include <sstream>
#include <string>
#include <iostream>
#include <opencv\highgui.h>
#include <opencv\cv.h>

using namespace cv;
//initial min and max HSV filter values.
//these will be changed using trackbars
int H_MIN = 0;
int H_MAX = 256;
int S_MIN = 0;
int S_MAX = 256;
int V_MIN = 0;
int V_MAX = 256;
//default capture width and height
const int FRAME_WIDTH = 640;
const int FRAME_HEIGHT = 480;
//max number of objects to be detected in frame
const int MAX_NUM_OBJECTS = 50;
//minimum and maximum object area
const int MIN_OBJECT_AREA = 20 * 20;
const int MAX_OBJECT_AREA = FRAME_HEIGHT*FRAME_WIDTH / 1.5;
//names that will appear at the top of each window
const string windowName = "Original Image";
const string windowName1 = "HSV Image";
const string windowName2 = "Thresholded Image";
const string windowName3 = "After Morphological Operations";
const string trackbarWindowName = "Trackbars";
void on_trackbar(int, void*)
{//This function gets called whenever a
	// trackbar position is changed




}
string intToString(int number){


	std::stringstream ss;
	ss << number;
	return ss.str();
}
void createTrackbars(){
	//create window for trackbars


	namedWindow(trackbarWindowName, 0);
	//create memory to store trackbar name on window
	char TrackbarName[50];
	sprintf(TrackbarName, "H_MIN", H_MIN);
	sprintf(TrackbarName, "H_MAX", H_MAX);
	sprintf(TrackbarName, "S_MIN", S_MIN);
	sprintf(TrackbarName, "S_MAX", S_MAX);
	sprintf(TrackbarName, "V_MIN", V_MIN);
	sprintf(TrackbarName, "V_MAX", V_MAX);
```

```
	//create trackbars and insert them into window
	//3 parameters are: the address of the variable that is changing when the
trackbar is moved(eg.H_LOW),
	//the max value the trackbar can move (eg. H_HIGH),
	//and the function that is called whenever the trackbar is moved(eg.
on_trackbar)
	//                                  ---->    ---->     ---->
	createTrackbar("H_MIN", trackbarWindowName, &H_MIN, H_MAX, on_trackbar);
	createTrackbar("H_MAX", trackbarWindowName, &H_MAX, H_MAX, on_trackbar);
	createTrackbar("S_MIN", trackbarWindowName, &S_MIN, S_MAX, on_trackbar);
	createTrackbar("S_MAX", trackbarWindowName, &S_MAX, S_MAX, on_trackbar);
	createTrackbar("V_MIN", trackbarWindowName, &V_MIN, V_MAX, on_trackbar);
	createTrackbar("V_MAX", trackbarWindowName, &V_MAX, V_MAX, on_trackbar);


}
void drawObject(int x, int y, Mat &frame){

	//use some of the openCV drawing functions to draw crosshairs
	//on your tracked image!
	//added 'if' and 'else' statements to prevent
	//memory errors from writing off the screen (ie. (-25,-25) is not within the
window!)

	circle(frame, Point(x, y), 20, Scalar(0, 255, 0), 2);
	if (y - 25>0)
		line(frame, Point(x, y), Point(x, y - 25), Scalar(0, 255, 0), 2);
	else line(frame, Point(x, y), Point(x, 0), Scalar(0, 255, 0), 2);
	if (y + 25<FRAME_HEIGHT)
		line(frame, Point(x, y), Point(x, y + 25), Scalar(0, 255, 0), 2);
	else line(frame, Point(x, y), Point(x, FRAME_HEIGHT), Scalar(0, 255, 0), 2);
	if (x - 25>0)
		line(frame, Point(x, y), Point(x - 25, y), Scalar(0, 255, 0), 2);
	else line(frame, Point(x, y), Point(0, y), Scalar(0, 255, 0), 2);
	if (x + 25<FRAME_WIDTH)
		line(frame, Point(x, y), Point(x + 25, y), Scalar(0, 255, 0), 2);
	else line(frame, Point(x, y), Point(FRAME_WIDTH, y), Scalar(0, 255, 0), 2);

	putText(frame, intToString(x) + "," + intToString(y), Point(x, y + 30), 1, 1,
Scalar(0, 255, 0), 2);

}
void morphOps(Mat &thresh){

	//create structuring element that will be used to "dilate" and "erode" image.
	//the element chosen here is a 3px by 3px rectangle

	Mat erodeElement = getStructuringElement(MORPH_RECT, Size(3, 3));
	//dilate with larger element so make sure object is nicely visible
	Mat dilateElement = getStructuringElement(MORPH_RECT, Size(8, 8));

	erode(thresh, thresh, erodeElement);
	erode(thresh, thresh, erodeElement);


	dilate(thresh, thresh, dilateElement);
	dilate(thresh, thresh, dilateElement);

}
void trackFilteredObject(int &x, int &y, Mat threshold, Mat &cameraFeed){

	Mat temp;
	threshold.copyTo(temp);
	//these two vectors needed for output of findContours
```

```
	vector< vector<Point> > contours;
	vector<Vec4i> hierarchy;
	//find contours of filtered image using openCV findContours function
	findContours(temp, contours, hierarchy, CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE);
	//use moments method to find our filtered object
	double refArea = 0;
	bool objectFound = false;
	if (hierarchy.size() > 0) {
		int numObjects = hierarchy.size();
		//if number of objects greater than MAX_NUM_OBJECTS we have a noisy
filter
		if (numObjects<MAX_NUM_OBJECTS){
			for (int index = 0; index >= 0; index = hierarchy[index][0]) {

				Moments moment = moments((cv::Mat)contours[index]);
				double area = moment.m00;

//if the area is less than 20 px by 20px then it is probably just noise
//if the area is the same as the 3/2 of the image size, probably just a bad filter
//we only want the object with the largest area so we safe a reference area each
//iteration and compare it to the area in the next iteration.
f (area>MIN_OBJECT_AREA && area<MAX_OBJECT_AREA && area>refArea){
					x = moment.m10 / area;
					y = moment.m01 / area;
					objectFound = true;
					refArea = area;
				}
				else objectFound = false;


			}
			//let user know you found an object
			if (objectFound == true){
				putText(cameraFeed, "Tracking Object", Point(0, 50), 2, 1,
Scalar(0, 255, 0), 2);
				//draw object location on screen
				drawObject(x, y, cameraFeed);
			}

		}
		else putText(cameraFeed, "TOO MUCH NOISE! ADJUST FILTER", Point(0, 50),
1, 2, Scalar(0, 0, 255), 2);
	}
}
int main(int argc, char* argv[])
{
	//some boolean variables for different functionality within this
	//program
	bool trackObjects = true;
	bool useMorphOps = true	;
	//Matrix to store each frame of the webcam feed
	Mat cameraFeed;
	//matrix storage for HSV image
	Mat HSV;
	//matrix storage for binary threshold image
	Mat threshold;
	//x and y values for the location of the object
	int x = 0, y = 0;
	//create slider bars for HSV filtering
	createTrackbars();
	//video capture object to acquire webcam feed
	VideoCapture capture;
	//open capture object at location zero (default location for webcam)
	capture.open(0);
```

```
	//set height and width of capture frame
	capture.set(CV_CAP_PROP_FRAME_WIDTH, FRAME_WIDTH);
	capture.set(CV_CAP_PROP_FRAME_HEIGHT, FRAME_HEIGHT);
	//start an infinite loop where webcam feed is copied to cameraFeed matrix
	//all of our operations will be performed within this loop
	while (1){
		//store image to matrix
		capture.read(cameraFeed);
		//convert frame from BGR to HSV colorspace
		cvtColor(cameraFeed, HSV, COLOR_BGR2HSV);
		//filter HSV image between values and store filtered image to
		//threshold matrix
		inRange(HSV, Scalar(H_MIN, S_MIN, V_MIN), Scalar(H_MAX, S_MAX, V_MAX),
threshold);
		//perform morphological operations on thresholded image to eliminate
noise
		//and emphasize the filtered object(s)
		if (useMorphOps)
			morphOps(threshold);
		//pass in thresholded frame to our object tracking function
		//this function will return the x and y coordinates of the
		//filtered object
		if (trackObjects)
			trackFilteredObject(x, y, threshold, cameraFeed);

		//show frames
		imshow(windowName2, threshold);
		imshow(windowName, cameraFeed);
		imshow(windowName1, HSV);


		//delay 30ms so that screen can refresh.
		//image will not appear without this waitKey() command
		waitKey(30);
	}

	return 0;
}
```

# Приложение Б

Отслеживание несколько объектов

```
#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/imgproc/imgproc.hpp>

#include<iostream>

////////////////////////////////////////////////////////////////////////////////
////////////
int main() {
	cv::VideoCapture capWebcam(0);			// declare a VideoCapture object and
associate to webcam, 0 => use 1st webcam

	if (capWebcam.isOpened() == false) {				// check if
VideoCapture object was associated to webcam successfully
		std::cout << "error: capWebcam not accessed successfully\n\n";// if not,
print error message to std out
		return(0);
				// and exit program
	}

	cv::Mat imgOriginal;	// input image
	cv::Mat imgHSV;
	cv::Mat imgThreshLow;
	cv::Mat imgThreshHigh;
	cv::Mat imgThresh;

	std::vector<cv::Vec3f> v3fCircles;			// 3 element vector of
floats, this will be the pass by reference output of HoughCircles()

	char charCheckForEscKey = 0;

	while (charCheckForEscKey != 27 && capWebcam.isOpened()) {		// until
the Esc key is pressed or webcam connection is lost
		bool blnFrameReadSuccessfully = capWebcam.read(imgOriginal);		//
get next frame

		if (!blnFrameReadSuccessfully || imgOriginal.empty()) {		// if frame
not read successfully
			std::cout << "error: frame not read from webcam\n";		//
print error message to std out
			break;
				// and jump out of while loop
		}

		cv::cvtColor(imgOriginal, imgHSV, CV_BGR2HSV);

		cv::inRange(imgHSV, cv::Scalar(0, 155, 155), cv::Scalar(18, 255, 255),
imgThreshLow);
		cv::inRange(imgHSV, cv::Scalar(165, 155, 155), cv::Scalar(179, 255, 255),
imgThreshHigh);

		cv::add(imgThreshLow, imgThreshHigh, imgThresh);

		cv::GaussianBlur(imgThresh, imgThresh, cv::Size(3, 3), 0);

		cv::Mat structuringElement = cv::getStructuringElement(cv::MORPH_RECT,
cv::Size(3, 3));

		cv::dilate(imgThresh, imgThresh, structuringElement);
```

```
            cv::erode(imgThresh, imgThresh, structuringElement);

            // fill circles vector with all circles in processed image
cv::HoughCircles(imgThresh, // input image
v3fCircles,// function output (must be a standard template library vector
CV_HOUGH_GRADIENT,// two-pass algorithm for detecting circles, this is the only choice
available
2,// size of image / this value = "accumulator resolution", i.e. accum res = size of
image / 2
imgThresh.rows / 4,// min distance in pixels between the centers of the detected
circles
100,// high threshold of Canny edge detector (called by cvHoughCircles)

50,// low threshold of Canny edge detector (set at 1/2 previous value)
10,// min circle radius (any circles with smaller radius will not be returned)
400);// max circle radius (any circles with larger radius will not be returned)

            for (int i = 0; i < v3fCircles.size(); i++) {// for each circle . . .

                  // show ball position x, y, and radius to command line
             std::cout << "ball position x = " << v3fCircles[i][0] // x position of
center point of circle
             << ", y = " << v3fCircles[i][1]// y position of center point of circle
             << ", radius = " << v3fCircles[i][2] << "\n";// radius of circle
            // draw small green circle at center of detected object
             cv::circle(imgOriginal,
                   // draw on original image
             cv::Point((int)v3fCircles[i][0], (int)v3fCircles[i][1]),// center point
of circle
             3,
                   // radius of circle in pixels
            cv::Scalar(0, 255, 0),
                   // draw pure green (remember, its BGR, not RGB)
                          CV_FILLED);
                   // thickness, fill in the circle
             // draw red circle around the detected object
                   cv::circle(imgOriginal,
                   // draw on original image
                   cv::Point((int)v3fCircles[i][0], (int)v3fCircles[i][1]),
             // center point of circle
                   (int)v3fCircles[i][2],
                   // radius of circle in pixels
                   cv::Scalar(0, 0, 255),
                   // draw pure red (remember, its BGR, not RGB)
                          3);
                   // thickness of circle in pixels
             }     // end fo
                   // declare windows
             cv::namedWindow("imgOriginal", CV_WINDOW_AUTOSIZE);// note: you can use
CV_WINDOW_NORMAL which allows resizing the window
             cv::namedWindow("imgThresh", CV_WINDOW_AUTOSIZE);// or CV_WINDOW_AUTOSIZE
for a fixed size window matching the resolution of the image

                   // CV_WINDOW_AUTOSIZE is the default

             cv::imshow("imgOriginal", imgOriginal);// show windows
             cv::imshow("imgThresh", imgThresh);

             charCheckForEscKey = cv::waitKey(1);// delay (in ms) and get key press,
if any
       }     // end while

       return(0);
}
```

# Приложение В

физической модель маятника

```
#include <opencv2\core\core.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\imgproc\imgproc.hpp>


#include <iostream>
#include <math.h>

#define PI 3.14159265

int main(int, char**)
{
	cv::VideoCapture cap(0); // open the default camera
	if (!cap.isOpened())  // check if we succeeded
		return -1;

	cv::Mat image_HSV;
	cv::Mat image_Color1;
	cv::Mat image_Color2;
	cv::Moments moments_color1;
	cv::Moments moments_color2;

	cv::vector<cv::vector<cv::Point> > contours_1;
	cv::vector<cv::vector<cv::Point> > contours_2;
	cv::vector<cv::Vec4i> hierarchy_1;
	cv::vector<cv::Vec4i> hierarchy_2;

	cv::Scalar color_1 = cv::Scalar(255, 51, 204);
	cv::Scalar color_2 = cv::Scalar(3, 249, 241);
	cv::Mat image_frame;
	cv::namedWindow("Angles", 0);

	std::string point1;
	std::string point2;


	cv::Point center_1;
	cv::Point center_2;
	for (;;)
	{

		cap >> image_frame; // get a new frame from camera
		if (image_frame.empty()) break;

		// Generates HSV Matrix
		cv::cvtColor(image_frame,   // Input image
			image_HSV, // output image in HSV
			CV_BGR2HSV); // constant refering to color space transformation

		//filtering image for colors
		cv::inRange(image_HSV, //input image to be filtered
			cv::Scalar(0, 155, 155), //min thresshold value
			cv::Scalar(18, 255, 255), // max threshold value
			image_Color1); //output image

		cv::inRange(image_HSV, //input image to be filtered
			cv::Scalar(144, 189, 0), //min thresshold value
			cv::Scalar(177, 255, 255), // max threshold value
			image_Color2); //output image
```

```
		/// Find contours
		findContours(image_Color1, // input image
			contours_1, // vector to save contours
			hierarchy_1,
			CV_RETR_TREE,
			CV_CHAIN_APPROX_SIMPLE,
			cv::Point(0, 0));

		findContours(image_Color2, // input image
			contours_2, // vector to save contours
			hierarchy_2,
			CV_RETR_TREE,
			CV_CHAIN_APPROX_SIMPLE,
			cv::Point(0, 0));


		// Get the moments
		cv::vector<cv::Moments> mu_1(contours_1.size()); // initialize a vector
of moments called mu, vector size the number of contours
		for (int i = 0; i < contours_1.size(); i++)
		{
			mu_1[i] = moments(contours_1[i], false);
		}

		cv::vector<cv::Moments> mu_2(contours_2.size()); // initialize a vector
of moments called mu, vector size the number of contours
		for (int i = 0; i < contours_2.size(); i++)
		{
			mu_2[i] = moments(contours_2[i], false);
		}

		///  Get the mass centers:
		cv::vector<cv::Point2f> mc_1(contours_1.size()); //vector to store all
the center points of the contours.
		for (int i = 0; i < contours_1.size(); i++)
		{
			mc_1[i] = cv::Point2f(mu_1[i].m10 / mu_1[i].m00, mu_1[i].m01 /
mu_1[i].m00);
		}

		cv::vector<cv::Point2f> mc_2(contours_2.size()); //vector to store all
the center points of the contours.
		for (int i = 0; i < contours_2.size(); i++)
		{
			mc_2[i] = cv::Point2f(mu_2[i].m10 / mu_2[i].m00, mu_2[i].m01 /
mu_2[i].m00);
		}


		/// Draw contours

		for (int i = 0; i< contours_1.size(); i++)
		{
			if (mu_1[i].m00>1000){
				center_1 = mc_1[i];

				drawContours(image_frame, contours_1, i, color_1, 2, 8,
hierarchy_1, 0, cv::Point());
				circle(image_frame, mc_1[i], 4, color_1, -1, 8, 0);

				//std::cout<< "red circle: " <<mc_1[i] << '\n';
			}

		}
```

```
		for (int i = 0; i< contours_2.size(); i++)
		{
			if (mu_2[i].m00>1000){
				center_2 = mc_2[i];
				drawContours(image_frame, contours_2, i, color_2, 2, 8,
hierarchy_2, 0, cv::Point());
				circle(image_frame, center_2, 4, color_2, -1, 8, 0);

				line(image_frame, center_2, center_1, color_2, 4, 8, 0);
				line(image_frame, center_2, cv::Point(image_frame.cols,
center_2.y), color_2, 4, 8, 0);


			}

		}
		std::cout << "Point 1" << center_1 << ",  Point 2" << center_2
			<< "  angle :" << atan((abs(center_1.y - center_2.y)*1.0 /
(center_1.x - center_2.x)*1.0))*(180.0 / PI)
			<< '\n';
		cv::imshow("Angles", image_frame);
		if (cv::waitKey(30) >= 0) break;
	}
	// the camera will be DE initialized automatically in Video Capture destructor
tan ( (abs(center_1.y-center_2.y)/abs(center_1.x-center_2.x))* PI / 180.0 )
	return 0;
}
```

# Приложение Г

Математической модель маятника

```
clc;clear all;
%The System parameters
m=0.5; b=0.006; L=0.08; g=9.81;
%For simplification
r=g/L;
k=b/(m*L);
% The Pendulum ODE
f= @(t,x) [x(2);-k*x(2) - r*sin(x(1))];

%Intial Condition
init=[pi/2; 0];

%Solve ODE the time interval set from 0 to 10 second
[t,x]=ode45(f,[0 200],init);

%%Animation

%Origin
O=[0 0];
axis(gca, 'equal');%Aspect ratio of the plot
axis([-0.09 0.09 -0.09 0.02]);%The limit of the plot
grid on;%Enable grid


%Loop for Animation
for i=1:length(t)
    %%mass point
    P=L*[sin(x(i,1)) -cos(x(i,1))];

    %Circle in orgin
    O_circ=viscircles(O,0.001);
    %Pendulum
    pend = line([O(1) P(1)],[O(2) P(2)]);
    %Ball
    ball=viscircles(P,0.005);

    %Time interval to update the plot
    pause(0.01);
    %Delete the previos object if is not the final loop
    if i<length(t)
        delete(pend);
        delete(ball);
        delete(O_circ);
    end
end
```