TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut
Tarkvaratehnika õppetool

Meditsiini proovide automaatne tuvastamine kasutades OpenCV tehisnägemise raamistiku

Magistritöö

Üliõpilane: Aleksandr Filonenko
Üliõpilaskood: 132382 IAPM
Juhendaja: Jekaterina Ivask

Tallinn
2015

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

(*kuupäev*) (*allkiri*)

Annotatsioon

Filonenko A.(2015) Meditsiini proovide automaatne tuvastamine kasutades OpenCV tehisnägemise raamistiku. Magistritöö. Tallinna tehnikaülikool.

Käesoleva töö eesmärgiks on anda lugejale ülevaade, kuidas saab lahendada automaatse tuvastamise probleemi reaalsete meditsiini proovide põhjal. Lisaks on käesoleva töö autor põhjalikult analüüsinud tehisnägemise põhimõisteid: pilt, värvi koordinaadid, pilditöötlus, pildi transformatsioon, OpenCV.

Olulisemad probleemid, mida autor töös püüdis lahendada, on järgmised: tõestada, et automaatset tuvastamise programmi saab teha odavalt, kvaliteetselt ja vähese inimressurssi abiga, kui kasutada avatud lähtekoodiga raamistiku OpenCV. Samuti vaatleb autor käesolevas töös probleemi, kuidas kirjutada automaatse tuvastamise algoritm algusest peale ja valida õiged pilditöötluse funktsioonid. Töö ja probleemid on käsitletud reaalse äri projekti põhjal ITBS OÜ ettevõttes, kus autor osales projekti tarkvara arendaja rollis.

Antud magistritöö tulemuseks on automaatse tuvastamise algoritm, mis võimaldab töötada meditsiini proovidega. Samuti on töös analüüsitud iga algoritmi sammu ja tehtud ettepanekud kuidas parandada kvaliteeti ja optimeerida programmi kiirust.

Lõputöö on kirjutatud vene keeles ning sisaldab teksti 74 leheküljel, 4 peatükki, 32 joonist, 5 tabelit.

Abstract

Filonenko A.(2015) Automatic recognition of medical samples using OpenCV computer vision framework. Master`s thesis. Tallinn university of technology.

The aim of this work is to give a presentation of how to solve automatic recognition problem based on real medical samples. Furthermore, the aim is to show and deeply analyze computer vision concepts, such as image matrix, color coordinates, image segmentation, image transforms and other image processing algorithms.

The main problem of this work is to prove that it is possible to build cheaply and highly qualified an automatic recognition program if to use open source computer vision framework named OpenCV. This problem has been solved in Estonian company ITBS OÜ and author was participating in that commercial project as a software developer.

The result of this work is a programing algorithm that could process medical samples, recognize them and calculate intensity based on predefined rules. In addition, every step of the algorithm was analyzed and suggested possible algorithm and speed optimizations.

The thesis is in Russian and contains 74 pages of text, 4 chapters, 32 figures, 5 tables.

Аннотация

Filonenko A.(2015) Автоматическое распознавание медицинских проб с помощью библиотеки компьютерного зрения OpenCV. Магистерская работа. Таллиннский технический университет.

Целью данной работой дать представление читателю о том, как можно решать задачи автоматического распознавания на основе реальных медицинских проб. В данной работе также представлены теоритические основы по таким основным понятиям компьютерного зрения, как изображение, цветовые координаты, сегментация, преобразования и обработка изображений.

Основной проблемой, которую решает эта работа является доказательство того, что можно сделать программу визуального распознавания дёшево, качественно и затрачивая мало человеческих ресурсов, если логично подходить к поставленной задачи и использовать библиотеку компьютерного зрения OpenCV с открытым исходным кодом. Это проблема решается на основе реального коммерческого проекта, разработанного в лаборатории фирмы ITBS OÜ.

Результатом данной работы является программный алгоритм автоматического распознавания, способный работать с медицинскими пробами. Также в работе проанализирован каждый шаг алгоритма и представлены предложения для улучшения и оптимизации скорости работы программы.

Данная работа написана на русском языке и содержит 74 страницы, 4 основных главы, 32 рисунка и 5 таблиц.

Терминология

Patch	***Patch*** Медицинская проба на бумажной основе, 0.5мм в диаметре
OpenCV	***Open source computer vision*** Библиотека компьютерного зрения с открытым исходным кодом
Spot	***Spot*** Контрольные точки на patch-е
Ключ	***Key*** Ключ, по которому определяется положение spot-ов на patch-е
Матрица	***Matrix*** математический объект, записываемый в виде прямоугольной таблицы, представляет собой совокупность строк и столбцов, на пересечении которых находятся элементы
Пиксель	***Pixel*** Наименьший логический элемент двумерного цифрового изображения
Маска	***Mask*** Маска с предполагаемыми местами spot-ов
Позитивный контроль	***Positive control*** Позитивные контроль patch-а, а именно 1 и 11 spot-ы, которые всегда имеют максимальную интенсивность
Негативный контроль	***Negative control*** Негативные контроли patch-а, а именно 3и 16 spot-ы, которые всегда имеют минимальную интенсивность
ROI	***Region of interest***

	Регион интереса на изображении - вырезанный из общего изображения кусочек искомого объекта
API	***Application programming interface*** интерфейс программирования приложений — набор классов, процедур, функций, структур и констант, предоставляемых приложением для коммуникации, используется внешними программами

Список изображений

Список таблиц

Оглавление

1. Введение

В современном мире существует массу открытых проблем в области медицины. Проблемы есть глобальные и локальные. К ряду глобальных относятся поиски методов лечения пока ещё неизлечимых болезней, к ряду локальных автоматизация процессов выявления заболеваний. Одну из таких локальных проблем автор решал исполняя рабочие обязанности разработчика программного обеспечения в ITBS OÜ предприятии.

Задачу, которую взялась выполнять фирма было создание прототипа по автоматическому визуальному распознавания медицинских проб в виде пластырей (дальше будут называться patch-ами). Непосредственная роль автора была в разработке программы визуального распознавания patch-ей, с помощью библиотеки компьютерного зрения с открытым исходным кодом OpenCV. Также в работе будет кратко представлена общая архитектура самого робота(сканера).

1.1 Проблема

Основанная проблема которую решает данная работа является автоматизация процесса распознавания результатов анализа. До производства автоматического робота, фирма выполняли анализ вручную на глаз с помощью микроспора. Данные полученные вручную варьировались в следующих промежутках: контрольная точка (дальше именуется как spot) не проявился, spot слабо проявился, spot явно проявился. Такие результаты не давали возможности проследить тенденцию развития. Отсюда у клиента появилась необходимость создать автоматический сканнер, который бы делал эту работу точнее и быстрее.

Данная работа является результатом двухлетней разработки алгоритма распознавания, которая распознаёт patch-и с точностью 90%. Работа является сборником теоритических и практических основ для написания программ автоматического визуального распознавания объектов. А также работа представляет интерес для фирмы разработчика ITBS OÜ, так как содержит полный анализ и документацию коммерческого проекта.

Практическая часть выполнялась в лаборатории фирмы ITBS OÜ, по адресу Raja 15, Tallinn.

1.2 Постановка задачи

Задача поставленная перед автором работы была создать алгоритм визуального распознавания patch-ей с помощью библиотеки с открытым исходным кодом OpenCV. Данный алгоритм должен распознавать более 95% всех patch-ей которые не были бракованы. Наглядный пример бракованных и набракованных patch-ей находится в Приложении 2. Программа должна отсеивать patch-и в которых отсутствуют 1 и 11 spot, а также в которых неровно вырезан ключ и боковые крылья. Программа должна автоматически отсеивать и испорченные patch-и.

Второй главной задачей является последовательная документация алгоритма распознавания. Так как программа собирает статистику, то администратору программы важно знать, из каких компонентов состоит алгоритм и как её настроить под определённый тираж patch-ей.

1.3 Методика

При выборе методики исходили из главного требования, прототип надо сделать быстро и дёшево. Поэтому была выбрана библиотека с открытым исходным кодом OpenCV. Проанализировав исходное изображение было выявлено, что основная фигура, которую надо распознать это круг. Так как в OpenCV во время пробных тестов функция CvHoughCircles показала отличный результаты, было решено остановиться на ней. Как альтернатива были испытаны разные сторонние методы и библиотеки по распознавания объектов.

Одной из альтернатив было OpenCV метод HaarTraining. К сожалению он не подошёл, одной из причин было то, что принтер не имеет возможности всегда печатать одинаковые spot-ы и их интенсивность обычно разная. Исходя из этого мы исключили возможность пользоваться этим методом. В третей главе будет подробно описан выбор технологий.

1.4 Обзор работы

Работа состоит из пяти основных частей: введение, теоритическая часть, практическая часть, результаты и заключение. В теоретической части будут представлены основы распознавания дигитальных изображений, описаны такие понятия как изображение, преобразования, сегментация, фильтрация и поиск объектов по изображению. В практической части будет представлен и пошагово расписан алгоритм распознавания поставленной задачи. Далее будут приведены результаты работы и сделаны выводы.

2. Теоретические основы

В этой главе будет представлены теоритические основы, на которых основана практическая часть работы. Первым делом будет представлена разница между человеческим зрением и компьютерным, далее опишется сама библиотека с открытым исходным кодом OpenCV, с помощью которой и написан алгоритм распознавания технического задания. Далее будет представлена теория сегментации, фильтрации и выделения контуров. Самыми последними будут главы поисков объектов в изображениях и наиболее популярные алгоритмы трансформации изображений.

2.1 Человеческое зрение

Человеческий глаз это оптический прибор. Этот прибор улавливает свет проходящий через линзу. Колбочки глаза являются сенсорами, которые чувствительный к свету. Существует три типа колбочек – зелёные красные и синие. Каждый цвет чувствителен к своей световой волне красный к красной зелёный к зелёной и синий к синей. Любой цвет, который вы видите, определяется тем, какое количество фотонов поглощают колбочки каждого из этих трех типов, и в каких именно пропорциях они поглощают эти фотоны. В камерах и других дигитальных устройствах тоже есть сенсоры, которые чувствительны к разным световым волнам, но они часто обладают другой спектральной гаммой. Поэтому каждый человек и каждая камера видят один и тот же объект по разному [3]. А соответственно компьютерное зрение видит это ещё иначе, отсюда и возникают сложности разработки алгоритмов, так как глазу виден контраст, например между объектом и фоном, в то время как компьютеру не видно, так как разница спектра например всего 1-2 единицы. Отсюда и появляется необходимость в расширении световой гаммы, использования много-битных фотографий, использования специальных фильтров и освещения, чтобы сделать изначальную фотографию максимально понятной для компьютерного зрения [1][2]. Пример камеры и 16 битных фотографий представлен в практической части в соответствии с нашим техническим заданием.

2.2 Компьютерное зрение

Компьютерное зрение –это технология создания искусственных систем, которые могут проводить обнаружение, слежение и классификацию объектов [8]. Область компьютерного зрения может быть охарактеризована как молодая, разнообразная и динамично развивающаяся, но в тоже время ещё не существуют чётких её границ, а также универсальных алгоритмов распознавания. На сегодняшний день большая часть задач решается индивидуального и эти методы трудно классифицировать и объединить в один алгоритм [9].

Примеры применения компьютерного зрения:

- **Медицина** – анализ видеоданных получаемых с помощью микроскопии, рентгенографии, ангиографии, ультразвуковых исследований и томографии. С помощью современных технологий из предоставленных данных можно обнаружить опухоли, проверить размеры органов и систему кровотока.

- **Промышленность** – контроль качества, проверка на наличие дефектов, проверка ориентации деталей.

- **Военная отрасль** – управление ракетами, беспилотными кораблями, обнаружение вражеской военной техники, датчики наблюдения и контроля территорий.

- **Транспортные средства** – автономные транспортные средства, включая подводные, наземные, воздушные и другие, использующие системы основанные на компьютерном зрении.

Далее рассмотрим типичные задачи, которые решает компьютерное зрение: распознавание, движение, восстановление сцены, восстановление изображений. Наиболее распространённой задачей является распознавание, которую я и буду применять на практике. Основные проблемы, которые решает распознавание следующие:

- Идентификация объектов

- Обнаружение

- Распознавание знаков
- Распознавание положения
- Поиск изображения по кусочку изображению

2.3 Обобщённый алгоритм компьютерного зрения

Как и в любой отрасли, так и в компьютерном зрении, существует обобщённый алгоритм, в нашем случае алгоритм распознавания. Он содержит в себе порядок действий который необходимо применить для обнаружения объекта по фотографии или видеозаписи. Конечно, в каждом конкретном случае некоторые пункты можно опустить, или добавить дополнительные, но общая структура практически всегда одинаковая. В практической части использовался частный случай этого алгоритма. Сам алгоритм и пошаговое описание представлен ниже [1]:

1. **Получение исходного изображения** – изображения получают с различных датчиков способных создавать 2D или 3D изображения или цепочку таких изображений. Изображения состоят из пикселей, количество пикселей зависит от свойств камеры. Значения пикселей обычно соответствуют интенсивности света в одной или нескольких спектральных полосах (цветные изображения или изображения в оттенках серого)

2. **Первичная обработка** – удаление шума, масштабирование, исправление контрастности, выделение ROI зон.

3. **Выделение деталей** – выделение деталей различного уровня сложностей, таких как линии углы, фигуры, границы и кромки.

4. **Сегментация** – выделения интересующих точек, деление изображения на нужные и ненужные объекты, разделение изображения на несколько сегментов по цветам, по классам объектов, по линиям [7].

5. **Классификация объектов** - классификация обнаруженного объекта по различным категориям.

2.4 OpenCV

OpenCV (Open Source Computer Vision) – библиотека алгоритмов компьютерного зрения с открытым исходным кодом. Распространяется под лицензии BSD и может свободно использоваться в академических и коммерческих целях. Реализована на C/C++, также разрабатывается для Python, Java, Ruby, Matlab, Lua и других языков программирования. Разработана компанией Intel Russia в Нижнем Новгороде и теперь поддерживается и развивается компаниями Willow Garage и Itseez. Библиотека кроссплатформенная, может работать на следующих операционных системах: Windows, Android, Maemo, iOS, BlackBerry, Linux и OS X. Далее будут представлены основные компоненты библиотеки и их свойства [5]:

- opencv_core — основная функциональность, математические функции, линейная алгебра
- opencv_imgproc — обработка изображений (фильтрация, преобразования.).
- opencv_highgui — упрощённый интерфейс пользователя
- opencv_ml — модели машинного обучения
- opencv_features2d — распознавание плоских объектов
- opencv_video — библиотека работы с видеоданными
- opencv_objdetect — обнаружение объёмных объектов
- opencv_calib3d — калибровка камеры, поиск стерео-соответствия и элементы обработки трехмерных данных.
- opencv_flann — библиотека быстрого поиска ближайших соседей
- opencv_contrib — текущий разрабатываемый код.
- opencv_legacy — устаревший код.
- opencv_gpu — CUDA функции для ускорения обработки за счёт видеокарты [5].

2.5 Изображение и свойства

Далее будут рассмотрены изображения, как основа компьютерного зрения, их виды и свойства. Будут показаны различные типы цветных изображений и основные их различия.

2.5.1 Основные определения

Представим изображение как матрицу *A*, имеющую *i* и *j* количество пикселей. Тогда элементом этой матрицы, то есть пикселем, пусть будет точка $\boldsymbol{a_{ij}}$.

Если пиксель $\boldsymbol{a_{ij}} \in \{0, 1\}$ принимает только два значения, то изображение называется **бинарным** и состоит только из чёрных и белых пикселей [4].

Если пиксель $\boldsymbol{a_{ij}} \in \{0, 1 \dots N-1\}$ принимает N значений, то изображение называется **полутоновым**(серого оттенка) и каждый пиксель может принимать N оттенков серого(градаций яркости) [4].

Если пиксель $\boldsymbol{a_{ij}} \in \left\{ \begin{pmatrix} x_{ij}^1 \\ x_{ij}^2 \\ x_{ij}^3 \end{pmatrix} \right\}$, где $x_{ij}^k \in \{1,\ 2, \dots N-1\}$, то изображение называется **цветным**, каждый пиксель может иметь любое из N^3 возможных значений цвета, определяемого соответствующими ему координатами $\left(x_{ij}^1, x_{ij}^2, x_{ij}^3\right)$ в цветовом пространстве [4].

Если пиксель $\boldsymbol{a_{ij}} \in \left\{ \begin{pmatrix} x_{ij}^1 \\ x_{ij}^2 \\ \vdots \\ x_{ij}^m \end{pmatrix} \right\}$, где $x_{ij}^k \in \{1,\ 2, \dots N-1\}$, то изображение называется **многоканальным**. Такое изображение состоит из совокупности M полутоновых изображений $\left\{x_{ij}^k\right\}$. Каждому пикселю соответствует M-компонентный вектор со значениями яркости, соответствующими ему во всех M изображениях [4].

2.5.2 Цветные изображения

Далее будут детально рассмотрены цветные изображения, так как именно они чаще всего использоваться в компьютерном зрении, в задачах визуального распознавания. Их массовое использование обусловлено тем что, они более информативны и содержат более детальные данные. Основным отличием цветных изображений является разность систем цветовых координат. Далее попробуем разобраться какие виды цветовых координат существует, поскольку правильный выбор цветовых координат может облегчить написание алгоритма распознавания или обработки изображений [4].

2.5.2.1 Система RGB

Колбочки человеческого глаза чувствительны к красным, синим и злёным волнам света. Система цветовых координат основанная на этих цветах называется RGB (Red, Green, Blue). Система RGB реализует аддитивный процесс цветосмешивания. Конечный цвет формируется путём смешивания различных пропорций цветов, соответствующих координатным цветам, попадающих в человеческий глаз без отражений [4].

Если интенсивность всех каналов цветов минимальная, получает чёрный цвет, когда максимальна – белый. Если все каналы имеют равное значение, получаем какой-то оттенок серого. Такая система цветовых координат самая распространённая и применяется во всех современных мониторах и телевизорах. Пример системы RGB представлен ниже [4].

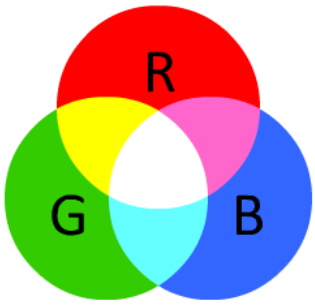


Рисунок 1. Система RGB

2.5.2.2 Система CMY и CMYK

Другой системой координат является система CMY, где C – Cyan(бирюзовый), M – Magenta(фиолетовый), Y – Yellow(жёлтый). Эта система субтрактивная и цвета используемые этой системой координат, называются вторичными цветами. Такая система координат используется в случае, когда изображение находится на прозрачном(плёнка) или непрозрачном носителе(бумага), и для рассматривания оно освещается белым светом. В таком случае нанесение красителя определённого цвета приведёт к тому, что из всего спектра падающего белого света будет вычитаться компонента, соответствующая цвету нанесённого красителя. В этом и состоит принципиальное отличие аддитивной и субтрактивной систем цветовых координат. Такая система применяться в принтерах. Однако непосредственное применение чистой системы CMY для принтеров имеет определённый недостаток, который обусловлен красителями при нанесении на бумагу. Дело в том, что максимальная интенсивность всех трёх компонентов должна приводить к получению чистого чёрного цвета. Однако из-за химического взаимодействия красителей между собой, которого невозможно избежать, смесь трёх компонентов обычно имеет не чисто чёрный цвет, а оттенок серого, обычно коричневатый. Для повышения качества распечатанных изображений было решено использовать отдельной компонентой чёрный цвет, что привело к появлению системы CMYK. Соответствующая величина четвёртой компоненты K – blacK(чёрный) применяется равной минимальному значению остальных компонентов C M и Y и вычитается из них. Такая система координат широко распространена во всех современных принтерах [4].

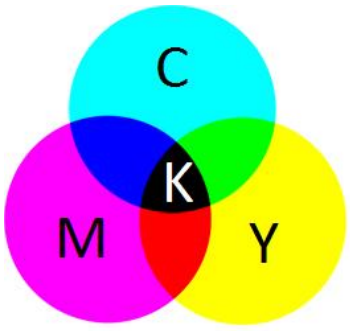


Рисунок 2. Система CMYK

2.5.2.3 Система HSV

Системы RGB и CMYK не являются естественными когда речь идёт о восприятии изображения человеком. Несмотря на то, что глаз человека имеет конструкцию соответствующую системы координат RGB, при оценки изображения человек использует другие параметры. Поэтому была создана система координат, координатные оси которой, соответствуют восприятию человеком цветного изображения. Такая ось HSV представлена ниже [4]:

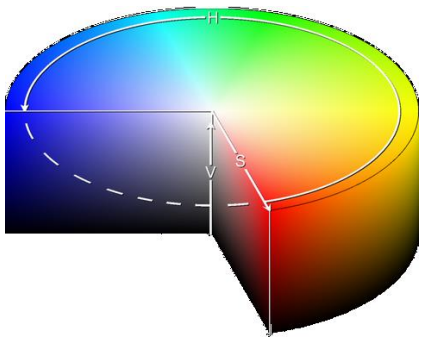


Рисунок 3. Система HSV[12]

В этой системы H – hue(тон), S – saturation(насыщенность), V – value (значение яркости).

- H – hue описывает основную длину волны, описывающую цвет. Варьируется в пределах 0—360°
- S – saturation показывает сколько чистого белого света содержится в цвете. Чем больше чистого белого света содержится в цвете, тем меньше насыщенность. Цвета, полученные в результате спектрального разложения белого света, не содержат чистого белого света совсем и следовательно, характеризуются максимальной насыщенностью (чистые цвета). Варьируется в пределах 0—100 или 0—1
- V – value мера яркости цвета. Также задаётся в пределах 0—100 или 0—1.

Эта модель является нелинейным преобразований модели RGB. Цвет, представленный в HSV, зависит от устройства, на которое он будет выведен [4].

2.6 Обработка изображений

Обработка изображений – это форма обработки данных представленных в виде фотографий или видеокадров. Изображения обрабатываются для подготовки к печати или транслированию, так и для распознавания текста, людей и других объектов, представленных на этих изображениях. Раньше, когда использовались аналоговые камеры, изображения обрабатывались изначально, то есть использовались разные оптические приборы и фильтры, которые применялись при съёмке или при фотографировании, сейчас в период цифровой обработки это делать не обязательно. Сейчас обрабатывать изображения стало ещё легче, удобнее и более качественней, так как появились программы, которые делают это автоматически, только лишь надо выбрать правильный метод или алгоритм. Конкретно в области компьютерного зрения, обработка изображений занимает наиважнейшую роль, так как именно на этой стадии происходят необходимые изменения фотографии для подачи её на вход алгоритма распознавания. Важно при обработке также соблюдать и порядок, например уменьшив размер изображения и выявив одни детали, в последующих шагах вам будет трудно определить другие детали, по –этому надо продумывать алгоритмы и порядок так, чтобы качество изображения при этом не портилось. Типичные задачи обработки изображений представлены ниже [2]:

- **Геометрические преобразования** - вращение и масштабирование.
- **Редактирование гистограммы** – расширение динамического диапазона гистограммы
- **Коррекция цветов** – изменения яркости и контраста
- **Коррекция дефектов** – редактирование и ретуширование
- **Интерполяция** – сглаживание
- **Наложение фильтров** – выделение определённых цветов или зон изображения

Многие из представленных выше методов мы будем использовать в нашей практической работе, некоторые из которых будут детально рассмотрены далее.

2.7 Гистограмма изображений

Гистограмма изображения – это график насыщенности изображения, суммарный или разделённый по каналам RGB. На основании этого графика можно определить экспозицию и контраст нашего изображения. В компьютерном зрении гистограмма изображений играет важную роль, так как с помощью гистограммы можно скорректировать входящее изображение с камеры, что в практической части нам и пригодится, а также усилить или расширить границу перехода цветов в представленном гистограммой диапазоне. Этот метод называется растягиванием гистограммы, задаётся нижний и верхний предел диапазона после чего все значения пикселей умножается на вычисленный коэффициент, после чего контраст между фоном и искомым объектом увеличивается. Малоиспользованные цвета просто пропадут и заменятся на белые пиксели, а наиболее часто используемые цвета будет разделены на множество других цветов, что придаст контраст деталям. Этот эффект применяется часто перед сегментацией, о которой будет детально расписано в следующей главе [1][2].

2.8 Сегментация

Сегментация изображений представляет собой разделение изображения на области по сходству познаков в их точках. В основном признаки разделяются на естественные и искусственные. Естественные признаки устанавливаются простым, а точнее визуальным анализом изображения, а искусственные – в результате специальной обработки и различных измерений. Более точно, сегментация изображений — это процесс присвоения таких меток каждому пикселю изображения, что пиксели с одинаковыми метками имеют общие визуальные характеристики [4].

Примерами естественных признаков являются: яркость, текстура, структура объекта. Примерами искусственных признаков являются: гистограмма, распределение яркости, спектр. Результатом сегментации изображения является множество сегментов, которые вместе покрывают всё изображение, или множество контуров, выделенных из изображения [4].

При анализе изображения необходимо его разбиение на области с целью измерения геометрических, топологических, спектральных и других характеристик областей, определяя их месторасположение, формы и геометрию. Основные простые виды

сегментации это сегментация по яркости, цветовым координатам, контурам и форме. Для сегментации изображений было разработано несколько универсальных алгоритмов и методов. Так как общего решения для задачи сегментации изображений не существует, часто эти методы приходится совмещать со знаниями из предметной области, чтобы эффективно решать эту задачу в её предметной области [4].

Далее будут представлены обобщённые алгоритмы сегментации и их описание:

- **Пороговая сегментация** – методы прямого деления на 2 или более области по признакам яркости, цветам или гистограммы.

- **Методы, основанные на кластеризации** – выбирается К центров кластера. Помешается каждый пиксель в кластер, с ближайшем к центру пикселем. Заново вычисляются центры кластера, усредняя все пиксели в одном кластере. Далее шаги повторяются до сходимости, пока не появятся сегменты.

- **Выделение краёв** – метод выделения объектов, основанный на алгоритмах, которые выделяют точки изображения, в которых резко изменяется яркость, есть другие виды неоднородностей или присутствуют резкие перепады значений цветовых координат.

- **Методы разреза графа** – сюда относятся такие популярные алгоритмы как изопериметрическое разделение, сегментация с помощью минимального остового дерева, нормализованные разрезы графов, случайное блуждание, минимальный разрез

- **Методы разрастания областей** - В качестве входных данных этот метод принимает изображений и набор семян. Семена отмечают объекты, которые нужно выделить. Области постепенно разрастаются, сравнивая все незанятые соседние пиксели с областью. Процесс продолжается, пока все пиксели не будут добавлены в один из регионов

- **Сегментация методом водораздела** - Вода, помещённая на любой пиксель внутри общей линии водораздела, течёт вниз к общему локальному минимуму яркости. Пиксели, от которых вода стекается к общему минимуму, образуют водосбор, который представляет сегмент

- **Многомасштабная сегментация** - Сегментация изображений выполняется в разных масштабах в масштабном пространстве, и иногда распространяется от мелких масштабов к крупным

2.9 Преобразования изображений

Преобразования изображений - это численные алгоритмы применяемые для извлечения элементов из изображений. В современном компьютерном зрении существуют масса алгоритмов, но в данной работе будет представлен именно тот, который будет применяться в практической части нашей работы, а именно алгоритм преобразования Хафа (Hough Transform). Так как в практической части мы будем извлекать объекты из изображений в виде кругов и овалов, мы сначала рассмотри алгоритм Хафа для выявления прямых, а потом версию алгоритма работающею с кругами.

2.9.1 Преобразования Хафа

Преобразование Хафа – это алгоритм применяемый для поиска объектов, относящихся к определённому классу фигур таких как прямых, кругов эллипсов. Алгоритм использует процедуры голосования [7].

Во многих случаях для выделение простых фигур из изображений, например таких как линии, круги и эллипсы используются алгоритмы сегментации, поиска краёв и границ. Однако, обычно из-за неравномерности или зашумлённости изображений, появляются пропуски или потерянные точки от кривой искомой фигуры. Также часто из-за искажений фотографии, угла камеры или других параметров искомые фигуры имеют отклонения от идеальной формы кругов, эллипсов и линей. Поэтому в таких случаях, помогает алгоритм Хафа – предназначенный решать проблему группировки граничных точек путём применения определённой процедуры голосования к набору параметризованных объектов изображения.

Если взять простейших случай, то алгоритм Хафа является линейным преобразованием нахождения прямых. Прямая может быть задана уравнением $y = mx + b,$ а значит может быть вычислена по паре точек (x, y) представленных на изображении. Идея алгоритма Хафа — учесть характеристики прямой не как уравнение, построенное по паре точек изображения, а в терминах её параметров, то есть коэффициента наклона m и точки пересечения с осью ординат b. Исходя из этого прямая, заданная уравнением

$y = mx + b$, может быть представлена в виде точки, с координатами (b, m) в пространстве параметров [29].

К сожалению, прямые параллельные осям координат имеют бесконечные значения для параметров m и b. Поэтому лучше описать уравнение в другом виде, в котором не возникают бесконечных параметров, а именно:

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{r}{\sin\theta}\right)$$

Где, r — это длина радиус-вектора ближайшей к началу координат точки на прямой, θ— это угол между этим вектором и осью абсцисс. После преобразования имеем следующие уравнение:

$$r = x\cos\theta + y\sin\theta$$

Отсюда теперь возможно связать с каждой прямой на изображении, в плоскости параметров, точку с координатами (r, θ), которая является уникальной при условии, что $\theta \in [0, \pi]$ и $r \in \mathbf{R}$, или что $\theta \in [0, 2\pi]$ и $r \geq 0$

Так как через каждую точку плоскости может проходить бесконечно много прямых, и если эта точка имеет координаты(x_0, y_0), то все прямые проходящие через неё, соответствуют уравнению:

$$r(\theta) = x_0 \cdot \cos\theta + y_0 \cdot \sin\theta$$

Это соответствует синусоидальной линии в пространстве Хафа (r, θ), которая, в свою очередь, уникальна для данной точки и однозначно её определяет. Если эти кривые или линии, соответствующие двум точкам, накладываются друг на друга, то точка представленная в пространстве Хафа, в том месте где они пересекаются, соответствует прямым на оригинальном месте, которые проходят через обе точки. Таким образом, проблема обнаружения коллинеарных точек может быть сведена к проблеме обнаружения пересекающихся кривых [6][29].

2.9.2 Нахождение кругов с помощью преобразования Хафа

Нахождение кругов работает таким же способом, как и нахождение прямых в преобразовании Хава, только для прямых мы использовали два параметра (r, θ), а для кругов нам надо использовать уже 3 параметра:

$$\mathrm{C}: (x_{center}, y_{center}, r)$$

Где (x_{center}, y_{center}) координаты центра круга, а r радиус круга [5].

3. Практическая часть

В этой главе будет детально представлено и проанализировано решение поставленного технического задания. Вначале опишется само задание и требования к нему. Туда входят требования по механическим и программным частям, минимальный коэффициент допустимых ошибок и требование к качеству всего решения.

Далее будет представлена механическая часть проекта, то есть сам сканнер и компоненты, такие как админ панель и веб интерфейс. Автор заверяет, что в разработке механической части он не участвовал, но с разрешения фирмы разработчика, представит его в своей дипломной работе для предоставления общей картины.

Далее будет пошагово и глубоко представлен и проанализирован алгоритм по распознаванию медицинских проб. Автор заверяет, что он принимал в разработке алгоритма непосредственно главную роль и представит исключительно свою работу.

3.1 Техническое задание

Далее будут продемонстрированы основные требования и задачи, которые должен решать сканнер. В список входят, как требования к сканеру, так и требования к алгоритму распознавания.

- Сканнер должен быть портативный, небольших размеров и с весом до 15кг
- Сборка аналогичного сканнера должна быть дешёвой
- Должен работать быстро, палетка 96 ячеек распознаваться до 1 часа.
- Иметь интерактивную и понятную веб админ панель, для анализа, архивации результатов и управления роботом.
- Алгоритм должен распознавать как минимум 95% patch-ей(бракованные patch-и не входят в это число)
- Алгоритм должен правильно определять сам patch и ключ

- Алгоритм находит позитивные контрольные spot-ы(1 и 11) и негативные(3 и 16)
- Алгоритм вычисляет интенсивность с учётом фона самого patch-а
- Алгоритм выделяет края спотов, если их край виден, в ином случае берёт spot в виде круга.
- Алгоритм автоматически отсеивает бракованные patch-и
- Алгоритм собирает статистику и представляет её пользователю
- Алгоритм можно подстраивать под определённый тираж печати patch-ей
- Изображения обрабатывается в 12-битовой системе
- Сканнер даёт возможность мануально распознавать patch-и в веб интерфейсе.

3.2 Сканер и компоненты

Далее будет представлен 3Д проект самого сканера, приведены компоненты, их наименования, и номера под которыми их можно найти на эскизе. Детальный эскиз внутренних модулей можно найти в Приложении 3.

1.	Материнская плата Mini ATX с дополнительным сетевым адаптером
2.	Микроконтроллеры: 2x Arduino Micro, 1x Arduino Nano
3.	LCD модуль для Arduino
4.	Индикаторные LED лампочки
5.	Импульсные кнопки
6.	Двухпозиционные микропереключатели
7.	Охлаждающие вентиляторы
8.	Моторы NEMA 17HS4417, 17HS0417

9.	Моторные приводы 2H MICROSTEP
10.	Жёсткий диск Kingston SSD 64GB
11.	Блок питания (PSU) IP-AD ITX
12.	Power over Ethernet (POE) D-Link DPE-101GI
13.	Камера BASLER Scan acA2500-14gc
14.	Линза BASLER Macro MLM-3XMP + Image Processing Lens 2/3" C 0.3X-1.0X F4.5-22C

Таблица 1. Список электронных частей сканера

15.	Червячные передачи
16.	Блок модуля камеры
17	Блок модуля подвижной платформы
18.	DRYLIN W одиночная линейная направляющая
19.	DRYLIN W двойная линейная направляющая
20.	Подшипник
21.	М6 стержень с резьбой
22.	Крепёжные и соединительные муфты
23.	Корпус из оргстекла
24.	Связующие внутренние запчасти напечатаны на 3D принтере из пластика ABS

Таблица 2. Список механических частей сканера

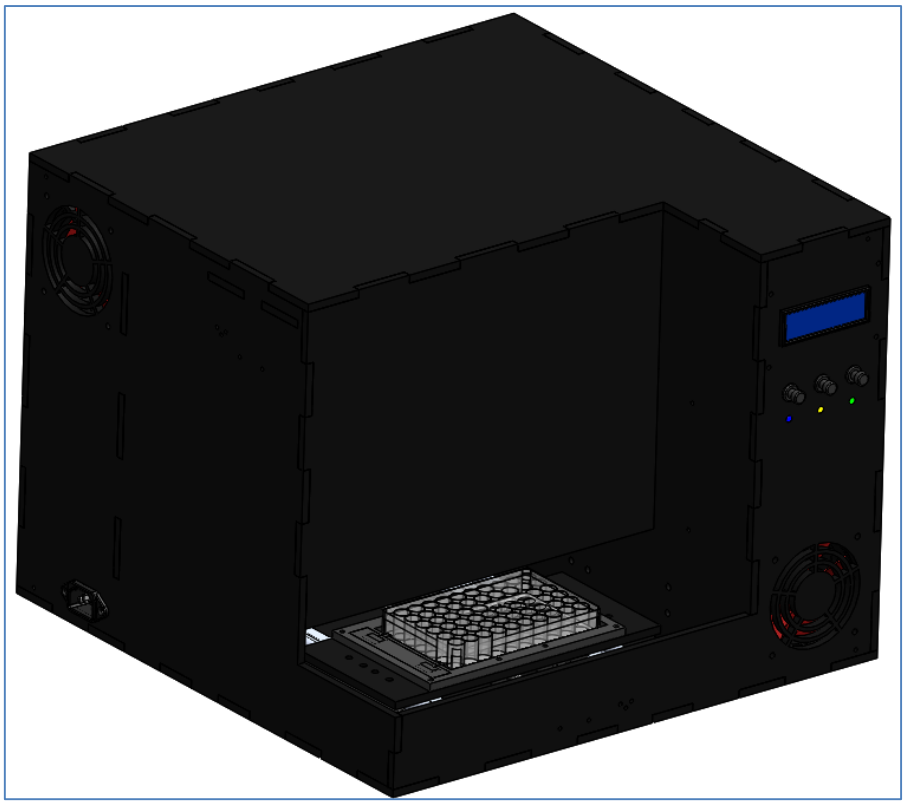


Рисунок 4. Внешний вид сканера

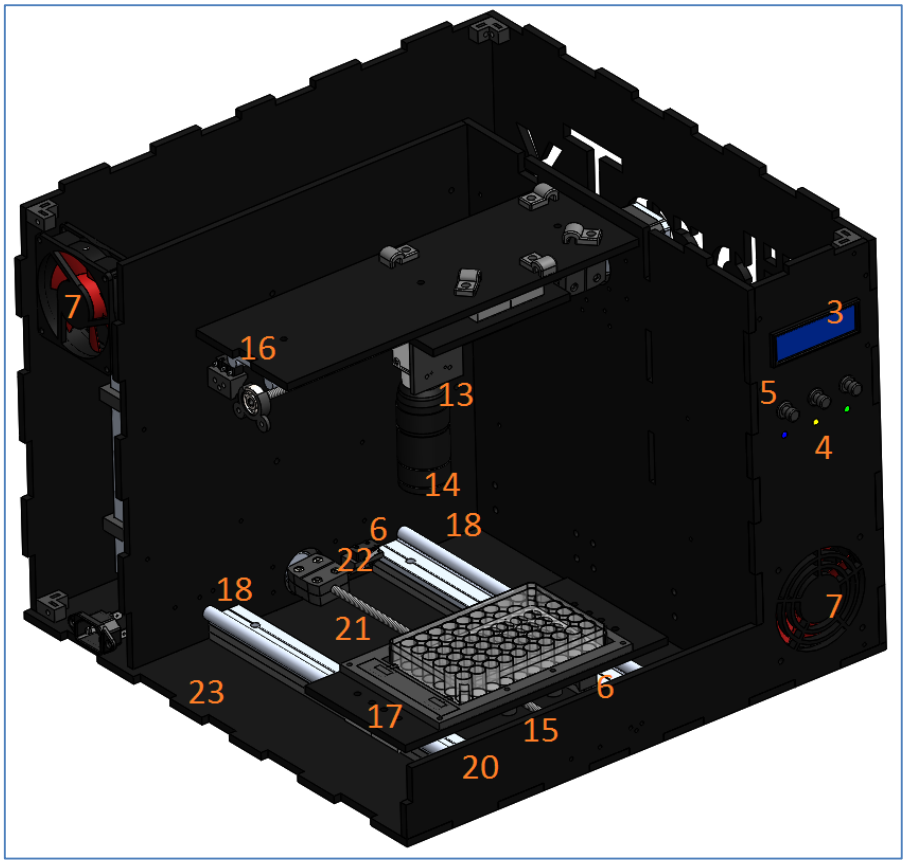


Рисунок 5. Внутренний вид сканера (часть 1)

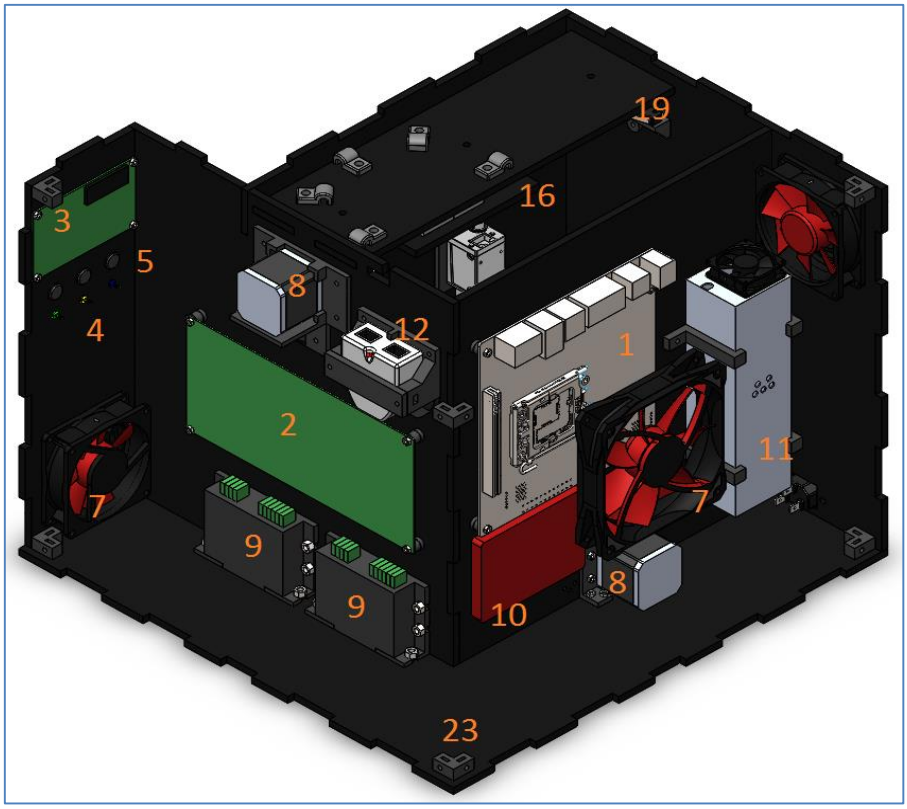


Рисунок 6. Внутренний вид сканера (часть 2)

Вторая важная, связующая часть, данного проекта является веб панель, в которой выполняются команды запуска, отдаются приказы моторам сканера и запускается алгоритм распознавания. Этот модуль написан на PHP языке программирования, отдельные скрипты запуска моторов, архивации, распознавания лежат на сервере, установленном внутри сканнера в виде BASH скриптов. В админ панели можно также воспользоваться ручным распознаванием, если patch был не распознан автоматический. Снимок экрана этой панели приведён ниже.

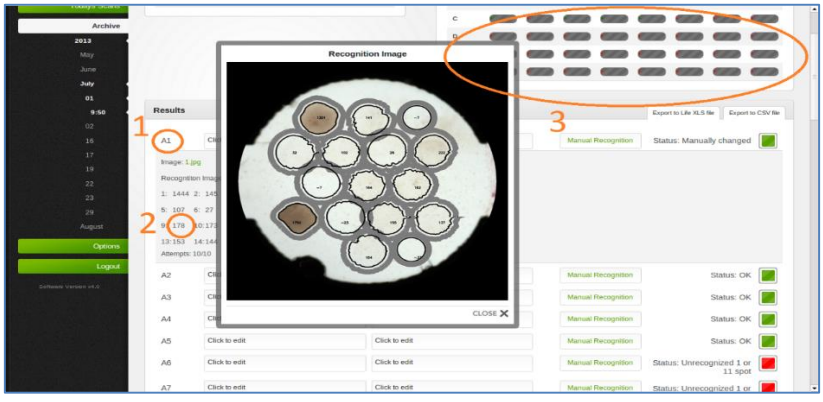


Рисунок 7. Админ панель сканера

3.3 Анализ выбора технологий для алгоритма распознавания

Так как существует большое количество технологий по компьютерному зрению, хотелось бы проанализировать наш выбор, а именно выбор Фреймворка OpenCV и метода преобразования Хафа(Hough transform) для нахождения кругов.

3.3.1 Выбор библиотеки компьютерного зрения

Основной выбор стоял между Фреймворками OpenCV, SimpleCV, PyCVF и Matlab. По нашем тестам OpenCV оказался быстрее всех, скорее всего по причине того, что написан на C++ и требовал мало вычислительных ресурсов. Так как компоненты сканера, а именно программируемые контроллеры Arduino, использовали C++, мы решили, что лучше делать весь проект на этом языке, так как любой человек из нашей команды мог бы подключится и помочь другому. Вторым немаловажным аргументом было то, что OpenCV распространяется под лицензией BSD, которая бесплатная и идеально подходит под наши коммерческие цели. Прототип конечно быстрее программировать на других языках, но зато на C++ можно использовать много функций по управлению памяти вручную. Также привлекает и то, что такие крупные компании как Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota используют OpenCV для разработки своих проектов. OpenCV один из наиболее популярных, быстроразвивающихся и современных Фреймворков, с отличной документацией и поддержкой.

3.3.2 Выбор алгоритма для нахождение кругов

Так как patch постоянно лежит перпендикулярно камере и имеет ярко выраженную форму круга, без искажений, то для нахождения patch-а и spot-ов было решено использовать линейные алгоритмы распознавания. Так как отсутствует возможность наклона камеры и искажения круга до овала, самым простым и проверенным способом было использовать преобразования Хафа, для нахождения кругов. Метод допускает неидеальные фигуры и отлично реагирует на различные входные параметры, в виде диаметров круга. Альтернативой было использование метода Haar training, но так как он основан на обучении, а количество тестовых проб было маленьким, метод постоянно путал ключ со spot-ом и patch-ем, потому что ключ имеет тоже форму полукруга. Разделить найденные объекты на классы было невозможным, из-за чего и пришлось отказаться от этого метода и выбрать преобразования Хафа.

3.4 Пошаговый алгоритм распознавания

В этой главе будет детально представлен алгоритм распознавания patch-ей. Название главы является шагом или функцией алгоритма. Каждая часть будет содержать описание самой функции и результат, который она возвращает в виде изображения. Сам программный код отдельных функций можно найти в Приложении 1.

В случае если хоть одна из функций представленных ниже не смогла пройти тело программы до конца, то она выбрасывает ошибку и указывает на бракованный patch. Примеры бракованных patch-ей, которые не подлежат распознаванию можно найти в Приложении 2.

3.4.1 Входящие изображение

Наиболее важную роль в результате распознавания играет входящая фотография. Важно, чтобы patch находился в центре, был равномерно освещён и не отражался в стенках палетки. Другим не менее важным свойством, которым должно обладать изображение – это качество. Наша камера Basler 2500 может снимать RGB 12-битовые фотографии. Фактически фотографии сохраняются в RGB 16-битовом формате, но реальной информации только12-бит, остальные ячейки заполнены нулями. В OpenCV мы побитово сдвигаем данные и получаем реальные значения в промежутке от 0 – 4095 (для каждого канала RGB), где 0 это полностью чёрный, 4095 это полностью белый. Пример входящей цветной фотографии из сканера приведён ниже.

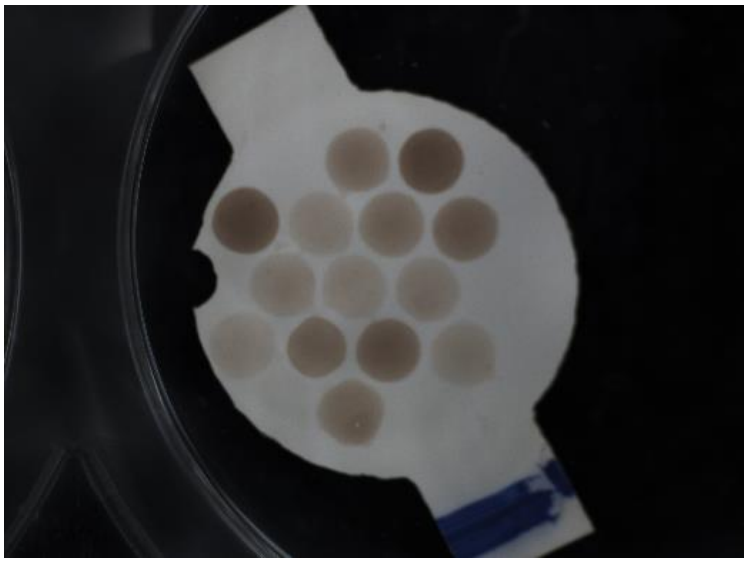


Рисунок 8. Входящие изображение 3 канала RGB 12 бит

Так как в результате опыта многие функции нашей программы дали наилучший результат при работе с чёрно-былыми фотографиями, такие как поиск кругов, поиск контуров и обрезание заднего фона, мы решили параллельно импортировать и чёрно-белое изображение, где значение всех трёх каналов RGB одинаковое. Получается, что мы имеем 2 матрицы параллельно проходящие цикл программы. По чёрно-белой матрице мы ищем объекты, по цветной мы высчитываем конечную интенсивность. Пример входящей чёрно-белой фотографии приведён ниже.

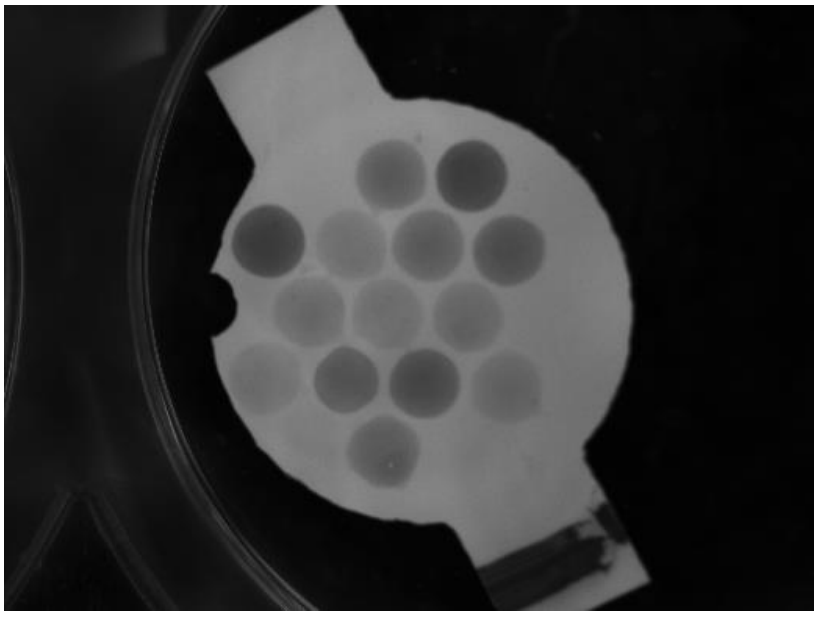


Рисунок 9. Входящее чёрно-белое изображение 12 бит

3.4.2 Фильтрация изображения

Следующим шагом нашего алгоритма является подготовка фотографии к поиску ключа, patch и spot-ов. Чтобы отсечь ненужные шумы, усилить выделение краёв, убрать визуальный мусор на заднем фоне и самом patch-е, необходимо проанализировать около 1000 фотографий и подобрать параметры, по которым можно от всех помех избавиться. Далее будут представлены методы, которые подошли именно нам, и параметры, с которыми они дают наилучший результат. Важно соблюдать порядок фильтрации так выходящая фотография одной функции является входящей для другой. Но бывает случается и так, что фильтрация не помогает, например если patch бракован, у него поцарапаны края или проделан лишний ключ или дырка на рабочей поверхности. В таких случаях алгоритм сбивается, но в конечном итоге он должен выдавать ошибку на следующих стадиях нахождения ключа или позитивных контролей. Примеры бракованных patch-ей можно найти в Приложении 2.

3.4.2.1 Фильтр инверсии

Суть фильтра инверсии найти одиночные пиксели, которые создают сильные шумы на заднем фоне patch-а. Такие пиксели перекрашиваются в белый цвет и потом при сегментации сливаются с другими аналогичными площадями пикселей. Фильтр был получен путём многих проб и алгоритм получился следующим: если значение голубого канала пикселя больше чем зелёного и красного, то такой пиксель мы перекрашиваем в белый цвет. Программный C++ код этой функции можно найти в Приложении 1. Результат который возвращает эта функция представлен ниже.

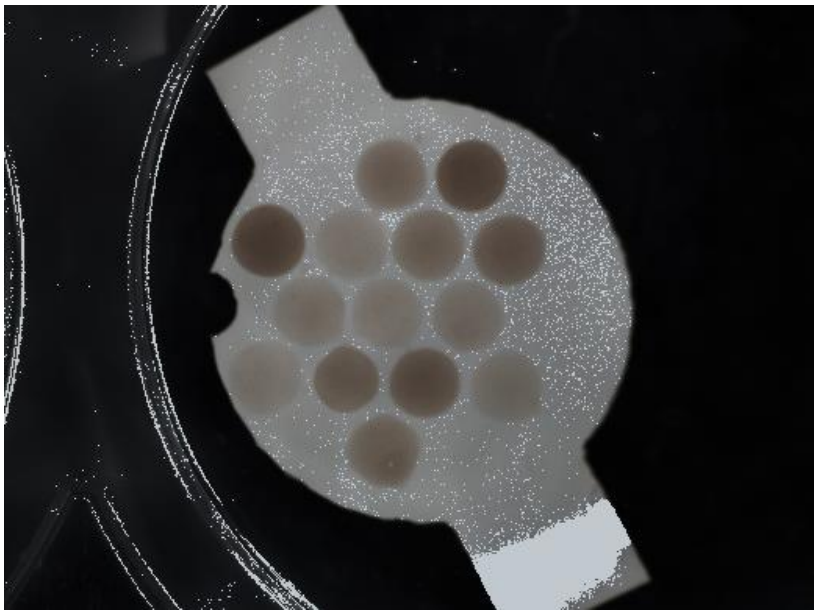


Рисунок 10. Результат инверсионного фильтра

3.4.2.2 Pyrmean сегментация

Следующим этапом фильтрации нашего изображения будет сегментация. Смысл этой функции состоит в том, чтобы сгладить цвета и у брать резкость пикселей. На вход идёт фотография с прошлого фильтра. Для сегментации мы используем функцию из OpenCV:

void **cvPyrMeanShiftFiltering**(const CvArr* src, CvArr* dst, double sp, double sr, int max_level=1)) [5].

- src – Входящая трёх канальная фотография.
- dst – Выходящая фотография того же формата и размера, то есть идентичная только пустая матрица.
- sp – Искомый радиус пространственного окна.
- sr – Искомый цветовой радиус.
- max_level – Максимальная степень пирамид для сегментации.

Параметры, которые были подобранны опытным путём для нашего алгоритма являются следующими: sp = **10**, sr = **40**, max_level = **5**.

Формула, по которой работает функция следящая: представим, что мы двигаемся по матрице слева на право и сверху вниз, тогда для каждого пикселя *x* и *y* и для каждого цвета RGB действует следующее правило [5].

$$(x,y) : X - \mathtt{sp} \leq x \leq X + \mathtt{sp}, Y - \mathtt{sp} \leq y \leq Y + \mathtt{sp}, ||(R,G,B) - (r,g,b)|| \leq \mathtt{sr}$$

Найденный средний пространственный радиус и средний цвет идёт в расчёт при следующей итерации [5].

$$(X,Y)\ (X',Y'), (R,G,B)\ (R',G',B').$$

Результат этой функции для нашего изображения представлен ниже:

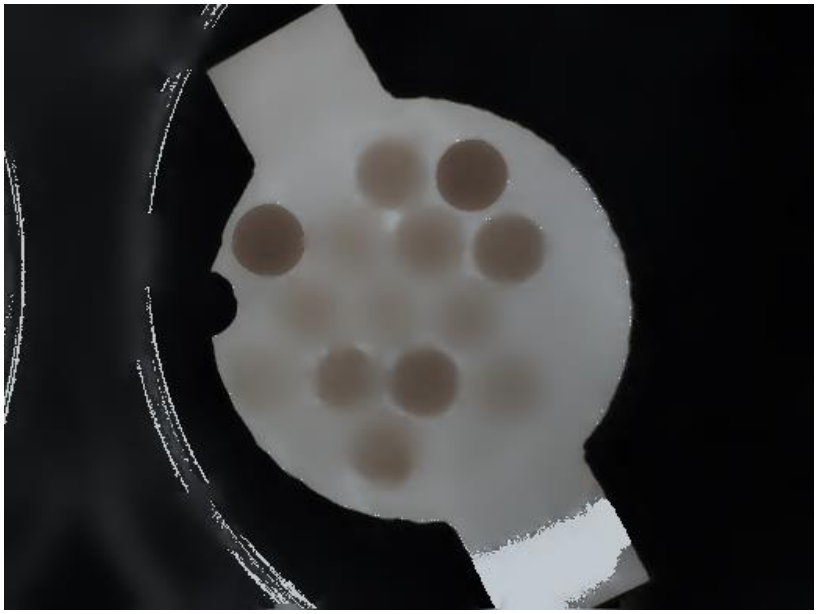


Рисунок 11. Результат Pyrmean сегментации

3.4.2.3 Пороговая сегментация

Основной принцип этой функции заключается в том, что всю фотографию надо поделить на два цвета чёрный и белый. Функция заимствована из OpenCV Фреймворка и работает следующим образом [5].

$$\mathtt{dst}(x,y) = \begin{cases} \mathtt{maxValue} & \text{if } \mathtt{src}(x,y) > \mathtt{threshold} \\ 0 & \text{otherwise} \end{cases}$$

Если пиксель больше порогового значения, то его красим в белый цвет, в другом случае в чёрный. Далее представим саму функцию и подобранные к ней параметры для

наших целей. После описания функции можно увидеть и результат работы этой функции, где на вход была подана фотография с предыдущего шага программы.

cvThreshold(const CvArr* src, CvArr* dst, double threshold, double maxValue, int thresholdType) [5]

- src – Входящая одно- канальная фотография.
- dst – Выходящая фотография того же формата размера, идентичная пустая матрица.
- threshold – значение порога(в нашем случае **40**)
- maxValue – значение, в которое перекрасим пиксель (в нашем случае **255**-былый цвет)
- thresholdType – тип сегментации(в нашем случае бинарный **CV_THRESH_BINARY**)

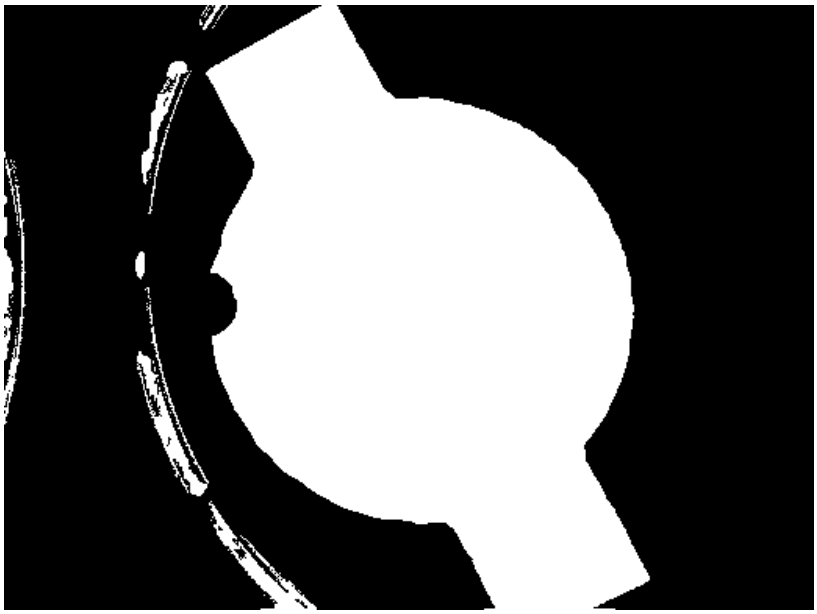


Рисунок 12. Результат пороговой сегментации

3.4.2.4 Фильтр для выделения контура patch-a

Последний фильтр данного цикла является фильтр выделения контуров. Так как для поиска patch-a мы используем функцию поиска кругов cvHoughCircles, а она показывает наилучший результат когда контур круга имеет ярко выраженную границу

в 1-5 пикселей, то для этих целей мы с помощью встроенных функций OpenCV: Canny, GuassianBlur, Sobel, ScaleABS будем выделять контур. Результат выходящий фотографии представлен ниже.

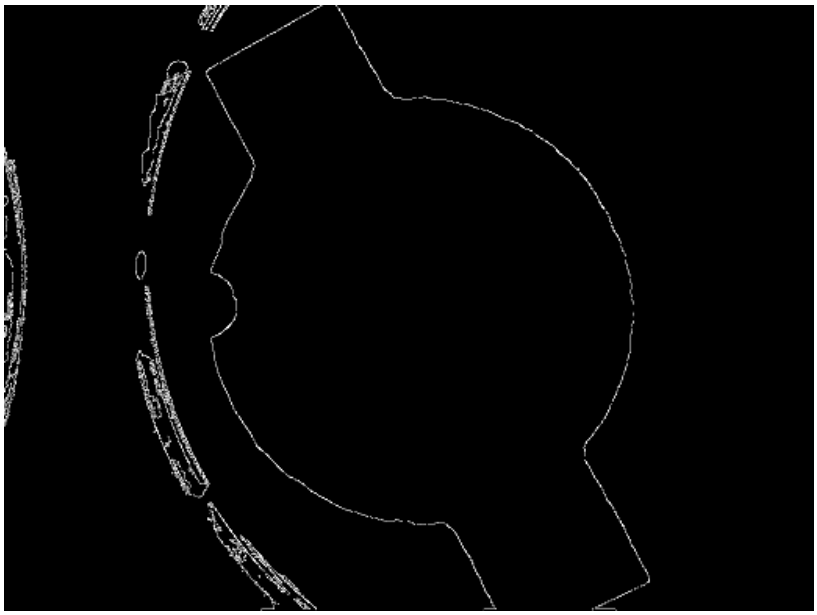


Рисунок 13. Результат фильтра для выделения контура patch-a

3.4.3 Поиск patch-a

Следующим шагом нашего алгоритма будет поиск самого patch-a. Patch будем искать по контуру из верхний фотографии с помощью функции cvHoughCircles. Так как нам нужен очень точный результат, мы будем минимизировать погрешность с помощью следующего свойства: мы будем вращать фотографию на один градус по кругу, каждый раз получая новую матрицу(фотографию) и далее будем находить круг, потом вычислять координаты относительно оригинального положения. По пришествию 360 градусов мы вычислим среднее арифметическое координат *x* и *y* и радиуса *r* и получим точный центр круга и его радиус.

После того как найдём patch, мы вырежем ненужную часть фотографии, чтобы уменьшить размер матрицы, тем самым ускорив алгоритм, который не будет тратить время на подсчёт ненужных пикселей, а также закрасим оставшуюся от patch-a площадь в чёрный цвет, то есть заполним матрицу нулями, чтобы при дальнейшем поиске нам эти пиксели не сбивали алгоритм.

Для поиска круга будем использовать следующую функцию, с вымеренными для нашего алгоритма параметрами, работа которой была представлена в теоритических основах:

cvHoughCircles(Mat& image, vector<Vec3f>& circles, int method, double dp, double minDist, double param1=100, double param2=100, int minRadius=0, int maxRadius=0) [5].

- image – входящая матрица 8 бит
- circles – вектор с найденными кругами
- method –метод **CV_HOUGH_GRADIENT**
- dp – обратное соотношение резолюции = **1**
- minDist – минимальное расстояния между искомыми кругами = **800**
- param1 – первый параметр hough-a= **1**
- param2 – второй параметр hough-a = **15**
- minRadius – минимальный радиус искомого круга = **680**
- maxRadius – максимальный радиус искомого круга = **700**

Далее будет представлен полный порядок действий алгоритма поиска patch-a и обрезания фона. Также будет представлен результат функции в виде фотографии.

1. Копия пустой матрицы
2. Вращение фотографии на 1 градус и поиск кругов с помощью cvHoughCircles функции
3. Конвертации координат в оригинальное положение матрицы и запись их в глобальный массив
4. Поиск среднего арифметического координат и радиуса
5. Вырезание всего, что находится далее чем на 10 пикселей от patch-a

6. Заполнение фона нулями (покраска в чёрный цвет)

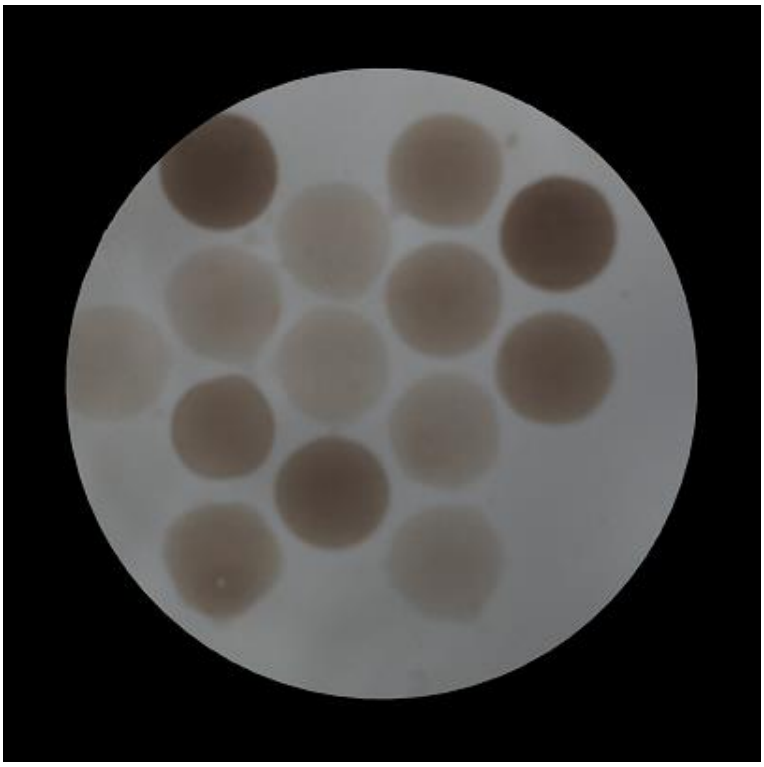


Рисунок 14. Patch после вырезания фона

3.4.4 Нахождение ключа

В этой главе рассмотрим алгоритм нахождения ключа. Ключ находится в левом нижнем углу patch-a, в виде полукруга. Ключ является индикатором положения patch-a. Чтобы понять какой spot какой по счёту, мы должны найти ключ и повернуть patch так, чтобы он был слева внизу, после чего можно применять маску для нахождения позитивных контролей. Детальный алгоритм с промежуточными изображениями представлен ниже:

1. Подаём на вход изображение после пороговой сегментации (Рисунок 9)

2. Конвертируем координаты центра и применяем их к входящему изображению

3. Уменьшаем радиус круга, то есть patch-a на 10 пикселей, это необходимо для уменьшения погрешности. Промежуточный результат представлен ниже

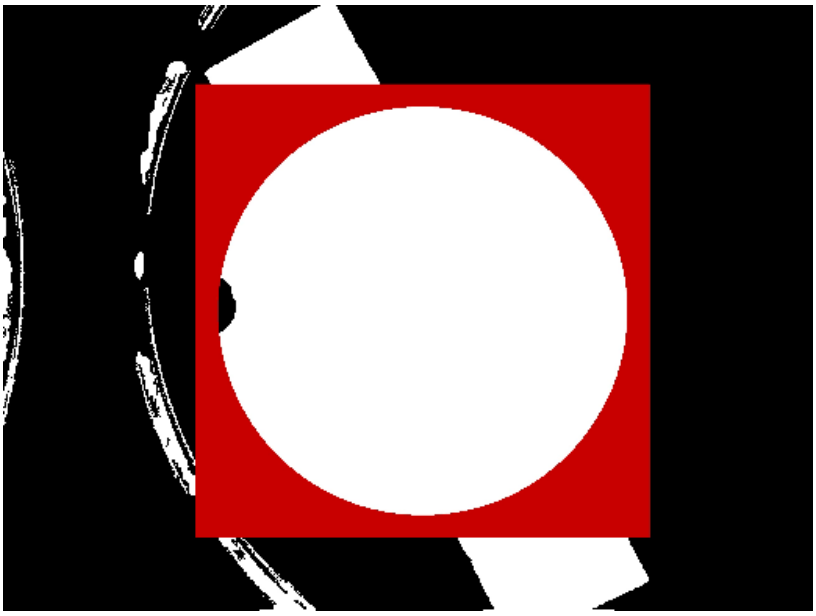


Рисунок 15. Нахождение ключа (часть 1)

4. Ищем все чёрные точки, которые находятся в замкнутом красном кругу, то есть проверяем каждый пиксель изображения, чтобы он был чёрным и подходил под формулу присутствия точки в круге представленную ниже. Координаты найденных точек складываем в глобальный массив.

$$(x-a)^2+(y-b)^2=r^2.$$

где *x* и *y* координаты центра окружности и *r* радиус этой окружности.

5. Далее находим среднее арифметическое точек *x* и *y* (зелёные точки) из собранного выше массива чёрных точек, что и является центром ключа (красная точка), с помощью которого мы сможем правильно повернуть patch.

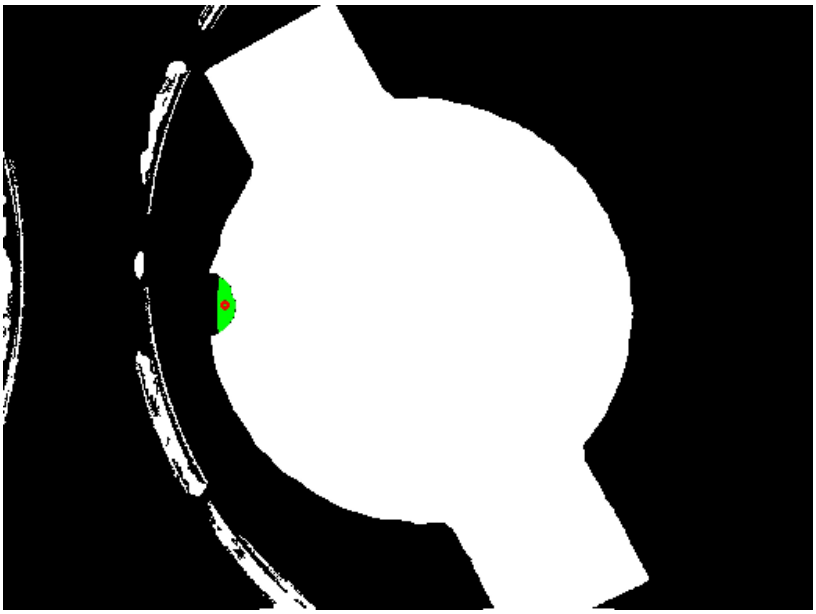


Рисунок 16. Нахождение ключа (часть 2)

3.4.5 Установка правильного положения patch-a

Вращение patch-a происходит с помощью функции OpenCV Фреймворка cvWarpAffine. Так как ключ должен находится не перпендикулярно основанию, а чуть левее, нам необходимо применить математику для вычисления правильного угла. Основная необходимость выравнивания patch-a является в том, чтобы расположить точки в правильном положение относительно горизонта или основания, что уменьшает погрешность при поиске кругов позитивных и негативных контролей. Неправильно повёрнутый patch может также отразиться и на подсчёте интенсивностей, если края spot-ов будут заходить на границы соседних spot-ов, то в конечном итоге при подсчёте интенсивностей фон будет браться из соседнего spot-a, что приведёт к полностью неправильным результатам. Поэтому вычисление угла и поворот необходимо произвести с точностью до 1 градуса.

Подробно метод вычисления угла представлен далее, а код находится в Приложении 1:

1. Имеем фотографию без заднего фона, угол прямого треугольника (константа 29.5 градусов), центр круга **CA** и координаты местоположения ключа **CB.**

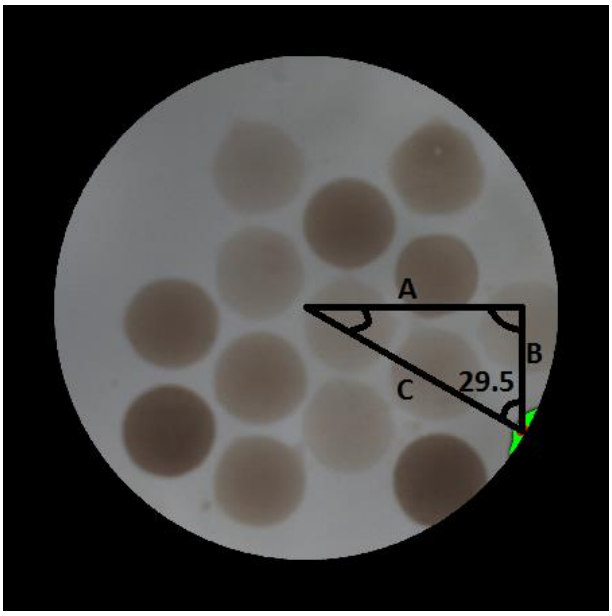


Рисунок 17. Установка правильного положения patch-a (часть 1)

2. Вычисляем точку **AB**, берём *x* координату из точки **CA** и *y* координату из точки **CB.**

3. Находим длины сторон **A** и **B** по математической формуле расстояния между двумя точками в двухмерном пространстве.

4. Проверяем в какой четверти находится ключ (I, II, III, IV)

5. Для I и IV вычисляем угол следующим образом: Угол = -((arctan(длина B/длина A))*180/число ПИ) -29.5

6. Для II и III вычисляем угол следующим образом: Угол = ((arctan(длина B/длина A))*180/число ПИ) -29.5

7. Если изображение перевёрнуто, добавляем к найденному углу ещё 180 градусов

8. Создаём матрицу `CV_32FC1` и вычисляем новую матрицу относительно центра patch-a, а не центра фотографии.

9. Поворачиваем фотографию с помощью функции cvWarpAffine на вычисленный угол. Результат правильно повёрнутой фотографии представлен ниже

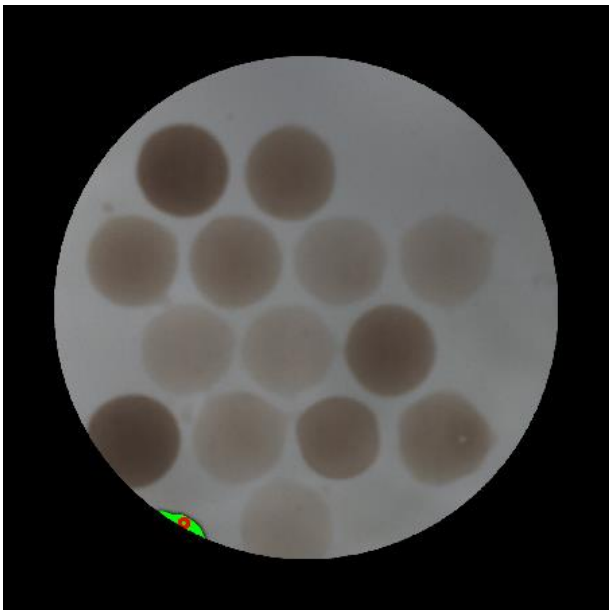


Рисунок 18. Установка правильного положения patch-а (часть 2)

3.4.6 Нахождение сдвига маски

В этой главе будет пошагово описан алгоритм поиска сдвига маски, то есть самих spot-ов, относительно найденного центра patch-а. Каждый раз принтер печатает patch-и и наносит биоматериал немного по-разному, что и вызывает смещение. Глазу это смещение практически не видно, но на patch-е с диаметром 0.5мм и под микроскопом оно может повлиять на весь результат распознавания. Зная тот факт, что позитивные контроли всегда проявлены и находятся на своих местах 1 и 11 ячеек маски, мы можем вычислить в какую сторону сдвинута сама маска и на сколько пикселей. Далее в следующих главах будет представлен алгоритм нахождения маски в порядке их выполнения.

3.4.6.1 Сжимание изображения

Так как края spot-ов иногда очень плохо видно, но в тоже время размер входящей фотографии 12 мегапикселей, нам позволяет сжать изображение в несколько раз тем самым выделив контуры 1 и 11 позитивных контролей. Этот метод используется только для нахождения контролей, при подсчёте интенсивностей полноразмерное изображение идёт в расчёт.

3.4.6.2 Адаптивная пороговая сегментация

Далее имея на руках сжатую фотографию мы прибегаем к следующему встроенному методу в Фреймворке OpenCV, который называется cvAdaptiveThreshold. Данный

метод сам подбирает наиболее удачный порог для деления фотографии на 2 цвета чёрный и белый, также он помогает нам выделить контур. Функция, подобранные параметры под наш алгоритм, и выходящий результат представлены ниже:

cvAdaptiveThreshold(const CvArr* src, CvArr* dst, double maxValue, int adaptive_method=CV_ADAPTIVE_THRESH_MEAN_C, int thresholdType=CV_THRESH_BINARY, int blockSize=3, double param1=5) [5].

- src – входящее изображение
- dst – выходящие изображение
- maxValue – максимальное используемого значение **CV_THRESH_BINARY = 255**
- adaptive_method – алгоритм сегментации = **CV_ADAPTIVE_THRESH_MEAN_C**
- threshold type = тип деления **CV_THRESH_BINARY**
- blockSize – Размер соседнего пикселя по которому вычисляется порог = **79**
- param1 – дополнительный параметр = 0

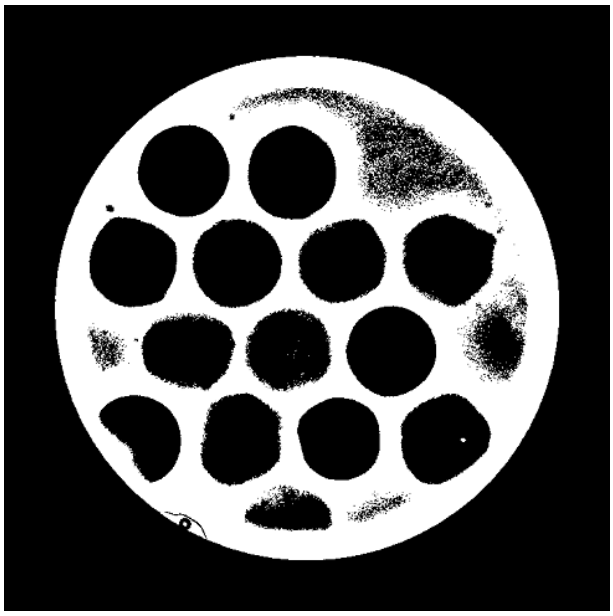


Рисунок 19. Результат функции cvAdaptiveThreshold

3.4.6.3 Фильтрация и усиление контуров spot-ов

Так как далее мы будем искать позитивные контроли с помощью фукнции cvHoughCircles, а она в свою очередь требует выявления контуров, то мы будем применять следующие фильтры: Canny, Sobel. По сути будем делать тоже самое, что и для нахождения patch-а только теперь применим это и для нахождения spot-ов. Результат этого метода представлен ниже:

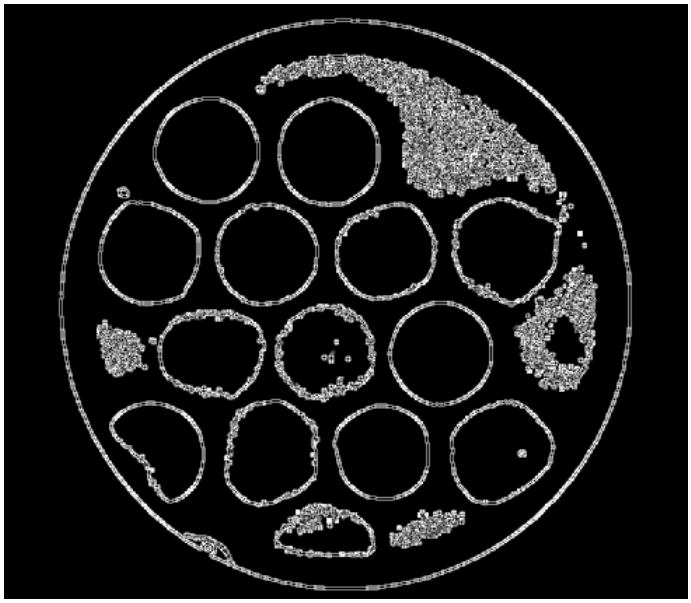


Рисунок 20. Усиление контуров spot-ов

3.4.6.4 Нахождение 1 и 11 spot-ов

Теперь приступаем к поиску 1 и 11 spot-ов. Применяем стандартную маску и в предполагаемой области начинаем поиск кругов с помощью функции cvHoughCircles. Опять применяем метод вращения фотографии вокруг центра и сбором координат в глобальный массив, после чего пересчитывается среднее арифметическое центра кругов и среднюю длину радиуса. Этот круг сравнивается с центром маски, а именно на сколько и в какую сторону он сдвинут относительно своей предполагаемой ячейки.

Другую немаловажную роль играет проверка расположения 1 и 11 spot-ов, которые называются позитивными контролями, так как они должны быть при любых условиях проявлены, и по ним можно сверить правильно ли повёрнут patch. Это делается с помощью подсчёта расстояния между точками и анализом координат центров. Результат нахождения позитивных контролей представлен ниже:

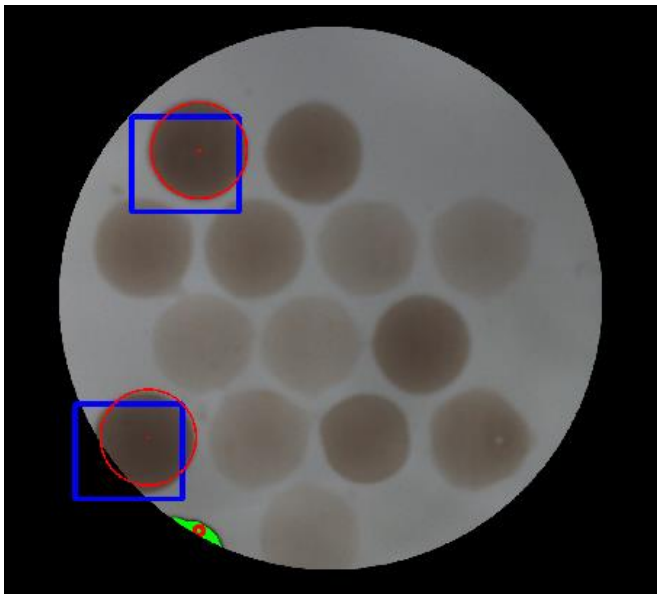


Рисунок 21. Нахождение позитивных контролей

3.4.6.5 Вычисление сдвига маски

Далее простым математическим путём вычисляем насколько и в какую сторону сдвинута маска: отнимаем от центра первой ячейки маски координату найденного контроля, тоже самое повторяем и со вторым контролем, далее вычисляем среднее арифметическое по оси *x* и *y*, направление определяем по знаку стоящему перед координатными. В следующих действиях мы будем приминать эти константы для выделения контуров spot-ов и подсчёта их внутренних и внешних интенсивностей.

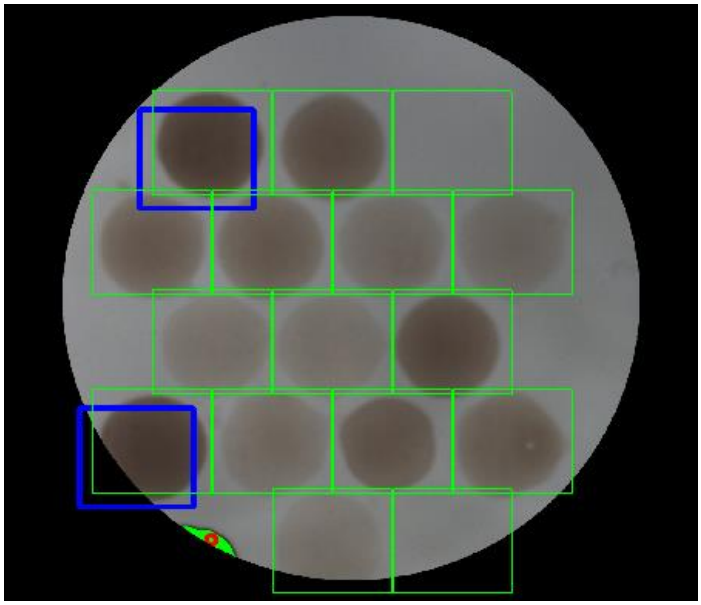


Рисунок 22. Вычисление сдвига маски

3.4.7 Нахождение spot-ов

В этой главе будет рассмотрен алгоритм нахождения spot-ов и их границ. Основной принцип, который здесь будет использован это нарезка фотографий, исходя из найденной позиции маски, на 16 маленьких фотографий. В каждой такой фотографии умещается целиком 1 spot и далее с ним происходят дополнительные обработки. Деление фотографии на маленькие кусочки даёт нам огромное преимущество, так как качество увеличивается, а процент ошибки при распознавании уменьшается, плюс значения соседних spot-ов и фон не влияют на подсчёт интенсивности искомого spot-а.

3.4.7.1 Вырезание ROI зон spot-ов

Далее приступаем к вырезанию 16 фотографий со spot-ами внутри. Найдя сдвиг маски мы прикладывает её и делаем вырез ROI(regioon of interest) зон прибавляя по 5 пикселей к каждому краю на случай погрешности в нахождении угла поворота и сдвига маски. Результат ROI зон представлен ниже.

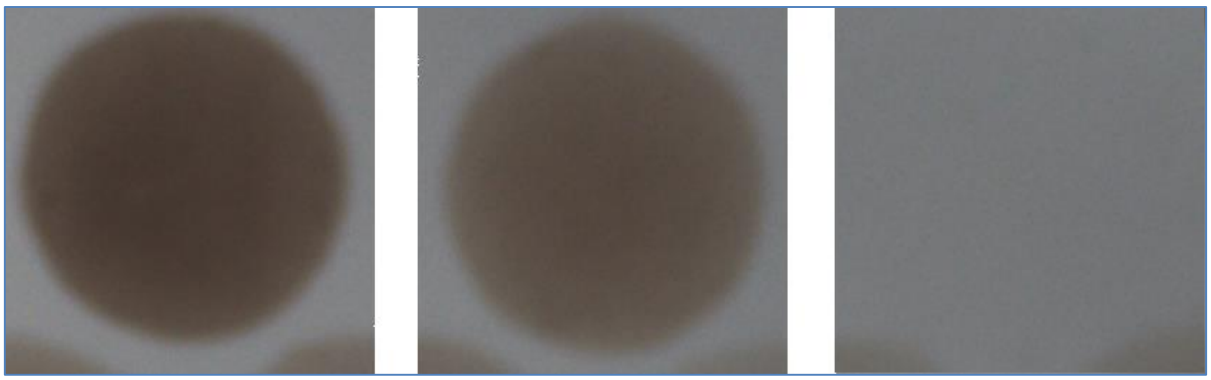


Рисунок 23. ROI зоны spot-ов

3.4.7.2 Растягивание гистограммы

Далее процесс идёт в цикле и выполняется для каждой маленькой фотографии отдельно. Следующим шагом нашего алгоритма является растягивание гистограммы для более чёткого выявления края spot-а. Алгоритм этой функции выглядит следующим образом:

1. Собираем значение RGB каждого канала и заносим в массив
2. Находим самое маленькое и самое большое значения по трём каналам, которые встречается не менее 5 раз в массиве

3. Вычисляем критические точки и коэффициент растяжения гистограммы. Принцип трасформации гистограммы представлен ниже:

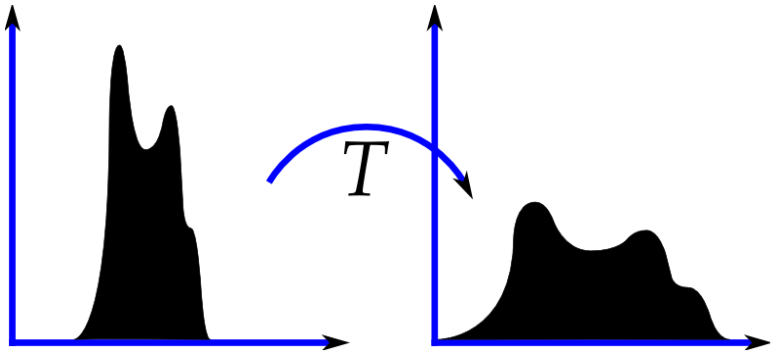


Рисунок 24. Принцип трансформации гистограммы[5]

4. Далее умножаем каждый пиксель и каждый канал RGB на найденный коэффициент. Получаем следующий результат:

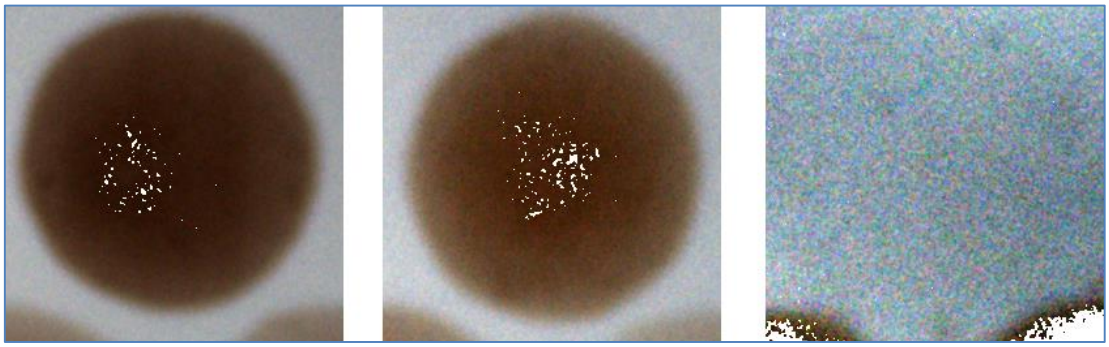


Рисунок 25. Результат трансформации гистограммы

3.4.7.3 Вычисление внутренней интенсивности spot-а

В следующем шаге нам надо найти внутреннюю интенсивность каждого spot-а. Эта интенсивность необходима для выявления контуров. Ставим точку в центр маленькой фотографии и берём результат средней интенсивности пикселей круга с радиусом 10 пикселей. Место на spot-е, которое мы берём в расчёт представлено ниже:

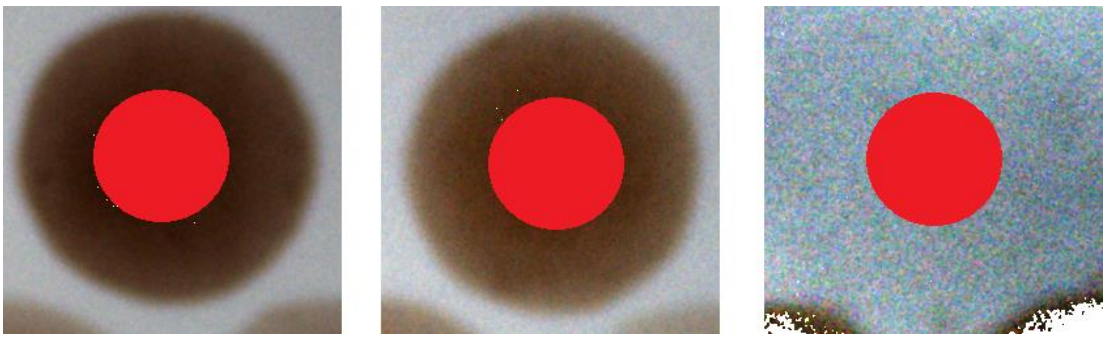


Рисунок 26. Подсчёт внутренних интенсивностей spot-ов

3.4.7.4 Фильтрация изображения

Исходя из найденных выше параметров мы применяем сегментацию для данной фотографии, чтобы выделить край чёрным а сам patch белым. Зная значения внутренней интенсивности мы проходя все пиксели проверяем, на сколько отличается значение зелёного канала, если больше 100 то окрашиваем пиксель в чёрной если меньше то в белый. Результат этого фильтра представлен ниже, теперь контур виден и его легко найти, код фильтра находится в Приложении 1, а результат ниже:

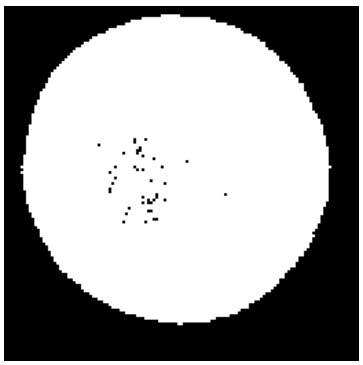


Рисунок 27. Сегментация ROI зоны

3.4.7.5 Выделение контура

Далее рассмотрим алгоритм выделения контура из Рисунка 24:

1. Находим круг с помощью расписанного ранее метода применяя функцию cvHoughCircles
2. Ставим ограничение, что край не может заходить за круг плюс 10 пикселей от найденного края круга
3. От центра двигаемся по кругу 360 к краю фотография, пока не упрёмся в чёрный пиксель
4. Если нашли пиксель сохраняем его в массив
5. Собираем все пиксели в цепочку и смотрим, чтобы не было скачков более 10 пикселей, если есть пропускаем средние значения при сохранении массива
6. Заносим точки контура в массив состоящий из координат *x* и *y*

Далее представлен результат этой функции, код находится в Приложении 1:

Рисунок 28. Контур spot-а

3.4.8 Подсчёт интенсивностей

В этой главе будет представлен алгоритм подсчёта интенсивностей. Он состоит из нескольких частей и имеет разные параметры и формулы для подсчёта разных групп spot-ов, а именно таких как позитивные и негативные контроли и результирующие spot-ы.

3.4.8.1 Формула

Формула подсчёта интенсивностей следующая:

Конечная интенсивность spot-а равна внутренней интенсивность spot-а минус интенсивность фона spot-а и минус средняя интенсивность негативных контролей (внутренняя интенсивность негативного контроля минус внешняя интенсивность негативного контроля).

Далее будет рассмотрены подсчёты отдельных частей формулы.

3.4.8.2 Негативные контроли

Негативные контроли это 3 и 16 spot-ы. Они никогда не определяются на изображении и вместо них ставится слепой круг со средним диаметром. Так как он равен фону, то и суммарная интенсивность фона минус найденной середины, должна быть равна нулю, но мы допускаем значения +- 50 единиц. Далее два этих spot-а обрабатываются и вычисляется средняя арифметическая интенсивность, которая и идёт в главную формулу. Пример слепого круга и интенсивности негативного контроля представлен ниже

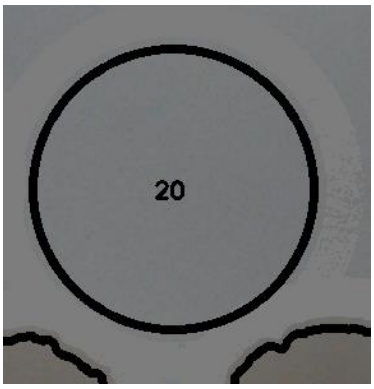


Рисунок 29. Spot с изломленным краем

В хорошо проявленных spot-ах, где нам удалось выявить край, мы подсчитываем интенсивность фона повторяя контур края, что даёт более точный результат. Пример представлен ниже:

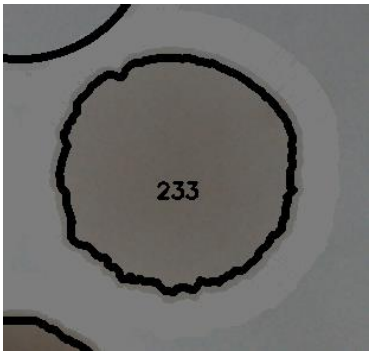


Рисунок 30. Подсчёт интенсивности фона spot-а с найденным краем

3.4.8.3 Круглые spot-ы

Если с помощью фильтра для выделения краёв нам не удалось извлечь сам край, то в таком случае мы ставим туда обычный круг найденный в промежуточных функция со средним радиусом. Для подсчёта интенсивности фона мы просто отступаем от края 10 пикселей и подсчитываем фон. Пример таких spot-ов представлен ниже:

Рисунок 31. Подсчёт интенсивности фона круглых spot-ов

3.4.9 Результирующая фотография

Чтобы пользователь мог визуально проверить, что всё распозналось правильно, ему предоставляется полноразмерная фотография, на которой правильно повёрнут patch, отмечены spot-ы, их края, если они присутствуют, и найдены интенсивности. По фотографии можно определить точность распознавания по позитивным и негативным контролям. Также на фотографии видно какую часть spot-а алгоритм считал внутренней интенсивностью, а какую фоном. В случае, если в результате визуальной проверки найдены отклонения, то patch можно обработать и в мануальном распознавании.

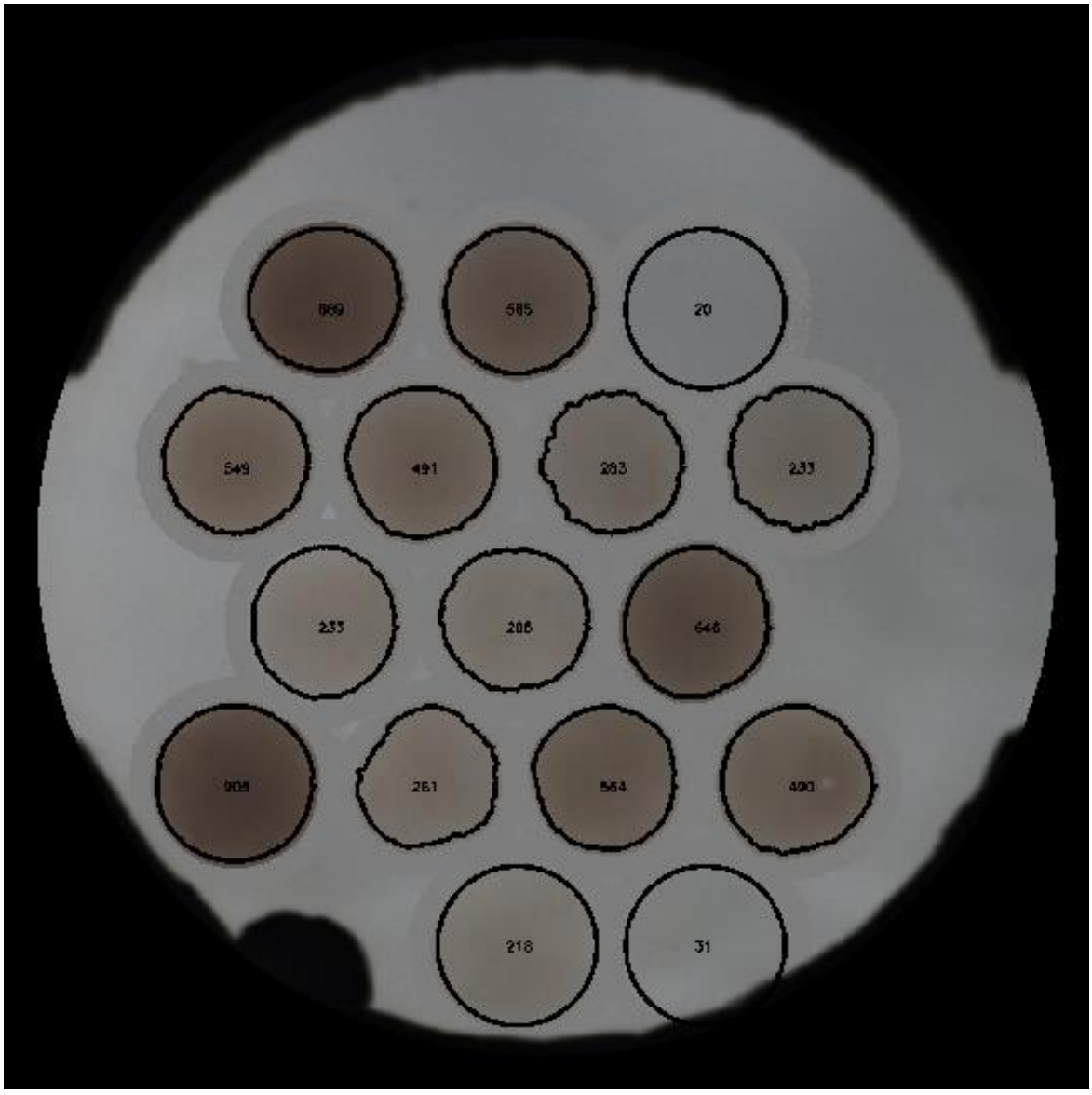


Рисунок 32. Визуальный результат распознавания алгоритма

3.4.10 Генерация CSV данных

Общение программы с главным веб интерфейсом происходит через API и формат передачи данных выбран CSV. Во время выполнения программа аккумулирует статистические данные, которые потом предоставляют возможность отследить параметры, ошибки, и проблемы прямо в самом веб интерфейсе. Пример того, что возвращает алгоритм представлен ниже, далее идёт описание представленных данных:

```
OK;1339;978;685;1;449;421;107;34205;15361;1754601137;3027;3585;144383759;2113;24271;914;889;1;20
OK;1339;978;685;2;709;421;100;31787;15361;1961323923;2810;3314;138748431;2200;24406;610;585;2;2.5
OK;1339;978;685;3;969;421;109;37737;15361;3436417178;2198;1705;119934463;2178;20850;20;20;3;0
OK;1339;978;685;4;319;641;94;27856;15361;2042838285;2568;3228;150548656;1994;23879;574;549;4;5
OK;1339;978;685;5;579;641;97;29954;15361;2182164911;2578;3250;150442340;2062;24665;516;491;5;10
OK;1339;978;685;6;839;641;89;25143;15361;1941712950;2487;2827;134772838;2179;23445;308;283;6;15
OK;1339;978;685;7;1099;641;95;28378;15361;2148254880;2518;3057;131289518;2260;23842;258;233;5;10
OK;1339;978;685;8;449;861;96;29222;15361;2550426117;2277;1859;150900078;2019;24226;258;233;6;15
OK;1339;978;685;9;709;861;95;28904;15361;2451475123;2329;2869;143337413;2099;23927;230;205;4;5
OK;1339;978;685;10;969;861;100;30833;15361;1797989888;2881;3280;137158657;2210;24243;671;646;2;2.5
OK;1339;978;685;11;319;1081;111;39138;15361;2033523584;3013;3241;141473136;2083;23430;930;905;1;20
OK;1339;978;685;12;579;1081;92;26922;15361;2208943615;2386;2875;137988782;2100;23047;286;261;6;15
OK;1339;978;685;13;839;1081;98;30027;15361;2023042955;2692;3275;141403359;2103;23659;589;564;5;10
OK;1339;978;685;14;1099;1081;97;29974;15361;2095411662;2639;3158;144465509;2124;24423;515;490;4;5
OK;1339;978;685;15;709;1301;109;37737;15361;3249345154;2302;2439;124756493;2059;20419;243;218;2;2.5
OK;1339;978;685;16;969;1301;109;33985;15361;3248539023;2104;2188;97170439;2073;16013;31;31;3;0
```

Таблица 3. CSV данные

1	Статус распознавания
2	ИД скана
3	День
4	Месяц
5	Год
6	Название входящего patch-а
7	Ошибки

8	Координата *x* центра найденного patch-a
9	Координата *y* центра найденного patch-a
10	Диаметр найденного patch-a
11	Название ячейки палетки
12	Номер угловой группы
13	Номер spot-a
14	Координата *x* центра найденного spot-a
15	Координата *y* центра найденного spot -a
16	Диаметр найденного spot -a
17	Количество пикселей в spot-e
18	Количество пикселей внутренней интенсивности
19	Сумма интенсивностей всех пикселей spot-a
20	Среднее арифметическое интенсивности spot-a
21	Сумма интенсивностей всех пикселей фона spot-a
22	Среднее арифметическое интенсивности фона spot-a
23	Количество пикселей взятых как фон spot-a
24	Финальная интенсивность spot-a без негативных контролей
25	Финальная интенсивность spot-a вместе с негативными контролями

Таблица 4. Параметры CSV данных

4. Результаты практической работы

Далее будут представлены результаты практической работы, а именно дана оценка работы программы и приведён график статистики распознавания. Затем будут представлены различные варианты оптимизации и ускорения алгоритма.

4.1 Оценка работы алгоритма распознавания

Для оценки работы алгоритма распознавания в программу был встроен модуль статистики, с помощью которого можно отследить погрешность распознавания и выявить процент удачных и неудачных попыток распознавать patch-и. Так как часто попадаются и бракованные patch-и, которые мы распознавать не должны, но должны давать сообщение об ошибки, то эти patch были исключены из статистики. Это означает что нашей задачей было привести программу к способности распознавать как минимум 95% набракованных patch-ей. Далее будет представлена таблица основанная на статистике распознавания patch-ей по версиям программы.

Версия	**Процент распознавания набракованных patch-ей**	**Время распознавания одного patch-а**
0.1	24%	9 секунд
0.8	45%	12 секунд
1.0	52%	33 секунды
2.0	85%	62 секунды
2.2	90%	71 секунда

Таблица 5. Результаты алгоритма распознавания

Из таблицы можно сделать вывод, что процент распознавания набракованных patch-ей постоянно растёт, но ещё не достиг необходимого минимума в 95%. Также с увеличением версии, то есть с добавлений в программу новых функций, постоянно увеличивается и время распознавания одного patch-а. Проблема времени начала играть большую роль, так как палетка в 96 patch-ей распознаётся 2 часа, что стало

неприемлемо для заказчика. Поэтому работа над алгоритмом продолжается постоянно, а конкретные шаги, которые уже сейчас предпринимаются будут представлены в следующей главе.

4.2 Будущие изменения алгоритма

Для улучшения качества распознавания и увеличения скорости программы в ближайшее время будут предприняты следующие шаги:

- Новое освещение – уменьшит отражение света от стенок палетки
- Замена палетки – планируется заменить палетку на матовую чёрную, что уменьшит неравномерное отражение света
- Алгоритм нахождения контуров – планируется улучшить алгоритм с помощью добавления новых промежуточных фильтров
- Оптимизация скорости – планируется ускорить выполнения программы с помощью многопоточности и возможности OpenCV выполнять расчёт с помощью видеокарты (CUDA).

Заключение

Количество задач автоматического распознавания в наши дни увеличивается особенно в таких огромных отраслях как медицина, промышленность, государственная безопасность и военная отрасль. Одну из таких задач компьютерного зрения удалось решить автору и описать в данной работе.

Целью данной работы было решение задачи распознавания медицинских проб с помощью компьютерного зрения, а также пошаговой документации алгоритма и анализом теоритических основ. Все вышеперечисленные цели осуществленный в полном объём.

Результатом данный работы является программный код, который применяется в коммерческих целях и был разработан автором исполняя рабочие обязанности в фирме ITBS OÜ в должности разработчика программного обеспечения. Так как из представленной выше статистики следует, что программа ещё не достигла необходимого минимума по успешному распознаванию медицинских проб, то работа над алгоритмом и его оптимизацией будет продолжатся и в будущем.

Kokkuvõtte

Tehisnägemise ülesannete arv on viimasel ajal kasvanud, eriti meditsiinis, tööstuses, riigi julgeolekus, sõjatööstuses ja teistes valdkondades. Üks sellistest tehisnägemise ülesannetest oli lahendatud ja analüüsitud käesoleva töö autori poolt.

Töö põhieesmärgiks oli algoritmi loomine, mis võimaldaks automaatselt tuvastada meditsiini proove, kasutades tehisnägemise raamistiku. Lisaeesmärgiks oli tuvastamise algoritmi dokumenteerimine ja teoreetiliste aluste analüüsimine. Kõik eelnimetatud eesmärgid olid saavutatud täies mahus.

Töö tulemuseks on programmi kood, mida kasutatakse ärilistel eesmärkidel. Kood on kirjutatud autori poolt ITBS OÜ jaoks, tarkvaraarendaja ametikohal. Kuna eespool välja toodud statistikast nähtub, et tuvastamise protsent on natuke väiksem kui nõutud miinimum, siis algoritmi programmeerimine ning selle optimeerimine jätkub ka tulevikus.

Summary

The number of needs of automatic recognition programs is increased, especially in vast fields such as medicine, industry, national security and military industry. One of such programs of computer vision was described and developed by the author.

The aim of this work is to solve a problem of automatic recognition of medical samples. In addition, it was necessary to analyze theoretical concepts and create a systematic documentation of algorithm.

The result of this work is a programming code that is used for commercial purposes and was designed by the author in ITBS OÜ Company in a role of software developer. Based on statistics it is showed that algorithm need additional development and speed optimization to fulfill the customer`s needs.

Использованная литература

[1] John C. Russ, The image processing handbook / John C. Russ – 6th ed, Boca Raton: CRC Press, 2011

[2] Gerhard X. Ritter, Joseph N. Wilson, Computer Vision Algorithms in Image Algebra, Иоса Raton: CRC Press, 2000

[3] Ioannis Pitas, Digital Image Processing Algorithms, Cambridge: Prentice Hall International(UK) Ltd, 1998

[4] С.В Абломейко, Д.М Лугановский, Обработка изображений: технология, методы, приминение, Минск: Амалфея, 2000

[5] OpenCV documentation, [Интернет материал]. Доступен: http://docs.opencv.org/ [Использовался 10.05.2015]

[6] Shapiro, Linda and Stockman, George. "Computer Vision", Prentice-Hall, 2001

[7] Duda, R. O. and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," Comm. ACM, 1992

[8] Tim Morris, Computer Vision and Image Processing. Palgrave Macmillan, 2004

[9] Л. Шапиро, Дж. Стокман. Компьютерное зрение = Computer Vision, Москва: Лаборатория знаний, 2006

[10] Дэвид Форсайт, Жан Понс. Компьютерное зрение. Современный подход = Computer Vision: A Modern Approach, Москва: Вильямс, 2004

[11] А.А. Лукьяница ,А.Г. Шишкин. Цифровая обработка видеоизображений, Москва: Ай-Эс-Эс Пресс, 2009

[12] [Интернет материал], изображение: http://image.acasystems.com/color-picker/faq-hsb-hsv-color-space.png

[13] J. R. Parker, Algorithms for Image Processing and Computer Vision (2nd ed.), Wiley, 2001

[14] Pedram Azad, Tilo Gockel, Rüdiger Dillmann, Computer Vision – Principles and Practice, Elektor International Media BV, 2008

[15] Nikos Paragios and Yunmei Chen and Olivier Faugeras, Handbook of Mathematical Models in Computer Vision, Springer 2005

[16] R. Fisher, K Dawson-Howe, A. Fitzgibbon, C. Robertson, E. Trucco, Dictionary of Computer Vision and Image Processing. John Wiley, 2005

[17] Richard Hartley and Andrew Zisserman, Multiple View Geometry in Computer Vision, Cambridge University Press, 2003

[18] William K.Pratt, Digital image processing, Wiley-Interscince, 2007

[19] Steve Mann, Intelligent image processing, Wiley-Interscience, 2002

[20] Andreas Koschan, Mongi Abidi, Digital color image processing, Wiley-Interscience, 2008

[21] Computer Vision Resource. [Интернет материал] Доступен: http://www.cvpapers.com/index.html [Использовался 10.04.2015]

[22] CVonline: The Evolving, Distributed, Non-Proprietary, On-Line Compendium of Computer Vision. [Интернет материал] Доступен: http://homepages.inf.ed.ac.uk/rbf/CVonline/ [Использовался 02.05.2015]

[23] Understanding the Hough transform. [Интернет материал] Доступен: http://matlabtricks.com/post-39/understanding-the-hough-transform [Использовался 09.05.2015]

[24] Hough Transform. [Интернет материал] Доступен: http://se.mathworks.com/help/images/hough-transform.html#buh9ylp-26 [Использовался 07.05.2015]

[25] Find circles using circular Hough transform. [Интернет материал] Доступен: http://se.mathworks.com/help/images/ref/imfindcircles.html [Использовался 08.05.2015]

[26] Histograms: Construction, Analysis and Understanding. . [Интернет материал] Доступен: http://quarknet.fnal.gov/toolkits/ati/histograms.html [Использовался 08.05.2015]

[27] Histogramm. [Интернет материал] Доступен: http://www.shodor.org/interactivate/activities/Histogram/ [Использовался 10.05.2015]

[28] Introduction to programming with OpenCV, [Интернет материал] Доступен: http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/opencv-intro.html . [Использовался 11.05.2015]

[29] Преобразования Хава, Википедия. [Интернет материал] Доступен: https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%A5%D0%B0%D1%84%D0%B0 [Использовался 11.05.2015]

[30] Сегментация изображений, Википедия. Доступен: https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%B3%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D0%B8%D1%8F_(%D0%BE%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%B8%D0%B7%D0%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) [Использовался 11.05.2015]

Приложение 1

```
IplImage* Invert(IplImage *tiff, IplImage *grey){

       int width     = tiff->width;
       int height    = tiff->height;
       int nchannels = tiff->nChannels;
       int step      = tiff->widthStep/ sizeof(unsigned short);

       unsigned short *data = ( unsigned short* )tiff->imageData;

       int r, g, b;

    for(int i = 0; i < height; i++)
    {
        for(int j =0; j < width ; j++)
        {
                    b = data[i*step + j*nchannels + 0];
                    g = data[i*step + j*nchannels + 1];
                    r = data[i*step + j*nchannels + 2];
                    if (b > r+1800 && b > g+1800){
                    CvPoint dsa = cvPoint(cvRound(j),cvRound(i));
                    cvCircle(grey, dsa, 1 ,cvScalar(206,200,193), 1);
                    }
        }
    }
       return grey;
}
```

```
IplImage* rotation(IplImage* img, IplImage* tiff) {

       int length = sqrt(pow((global.x - key.x), 2) + pow((global.y - key.y), 2));
       if( length < 550 ) {
              exit(1);
       }
       CvPoint rec_center = cvPoint(key.x, key.y);
       CvPoint third= cvPoint(global.x,rec_center.y);
       double  length_a, length_b;
       if(global.y  >= third.y) {
              length_a = global.y - third.y;
       }
       else{
              length_a = third.y - global.y;
       }
       if(global.x >= rec_center.x) {
              length_b = global.x - rec_center.x;
       }
       else{
              length_b = rec_center.x - global.x;
       }

       double angle;
       //Angle calculation, 1 "if" for I nad IV quater, 2 "if" for II and III quater
       if((rec_center.x >= global.x && rec_center.y >= global.y) || (rec_center.x <=
```

```
global.x && rec_center.y <= global.y)){
		angle = -29.5-((atan(length_b/length_a))*180/3.1415926);
	}
	else{
		angle = -29.5+((atan(length_b/length_a))*180/3.1415926);
	}
	//Adding 180 degress to angle if picture is up side down
	if( rec_center.y < global.y){
		angle = angle + 180;
	}

	cvCircle(img, global, cvRound(Global_radius+125), cvScalar(0, 0, 0), 250);
	cvCircle(img, global, cvRound(Global_radius+375), cvScalar(0, 0, 0), 250);

	cvSetImageROI(img,cvRect(global.x - 750, global.y - 750, 1500, 1500));
	IplImage * tmp = cvCreateImage(cvGetSize(img), img->depth, img->nChannels);
	cvCopy(img, tmp);

	cvSetImageROI(tiff,cvRect(global.x - 750, global.y - 750, 1500, 1500));
	IplImage * tmp2 = cvCreateImage(cvGetSize(tiff), tiff->depth, tiff->nChannels);
	cvCopy(tiff, tmp2);

	IplImage *rotated = cvCloneImage(tmp);
	CvMat* rot_mat = cvCreateMat(2, 3, CV_32FC1);
	double scale = 1.0;
	CvPoint2D32f center = cvPoint2D32f(tmp->width / 2, tmp->height / 2);
	cv2DRotationMatrix(center, angle, scale, rot_mat);
	cvWarpAffine(tmp, rotated, rot_mat);

	supertiff = cvCloneImage(tmp2);
	CvMat* rot_mat2 = cvCreateMat(2, 3, CV_64FC1);
	CvPoint2D32f center2 = cvPoint2D32f(tmp2->width / 2, tmp2->height / 2);
	cv2DRotationMatrix(center2, angle, scale, rot_mat2);
	cvWarpAffine(tmp2, supertiff, rot_mat);

	return rotated;
}
```

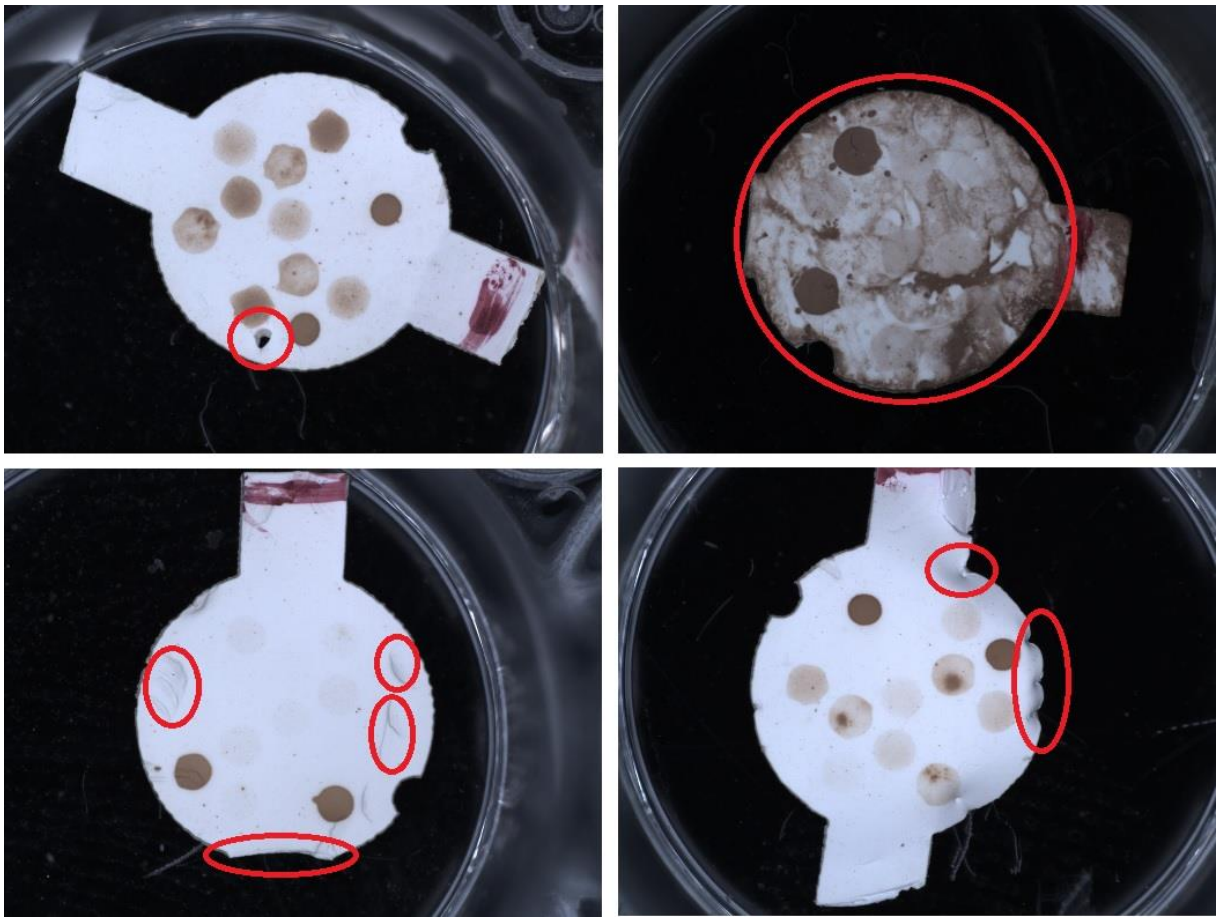
```
IplImage* edgeFilter(IplImage *tiff, IplImage *eightbit, int num){

	int width     = tiff->width;
	int height    = tiff->height;
	int nchannels = tiff->nChannels;
	int step      = tiff->widthStep/ sizeof(unsigned short);
	unsigned short *data = ( unsigned short* )tiff->imageData;
	int r, g, b;
	int count = 0;
	CvPoint pic_center = cvPoint(tiff->width / 2, tiff->height / 2);

    for(int i = 0; i < height; i++)
    {
	 for(int j =0; j < width ; j++)
	 {
			CvPoint dsa = cvPoint(cvRound(j),cvRound(i));
			b = data[i*step + j*nchannels + 0];
			g = data[i*step + j*nchannels + 1];
			r = data[i*step + j*nchannels + 2];
			if(b > (blue[num] + 6500)){
				data[i*step + j*nchannels + 0] = 0;
				data[i*step + j*nchannels + 1] = 0;
				data[i*step + j*nchannels + 2] = 0;
```

```
                    }
                    else{
                           data[i*step + j*nchannels + 0] = 65280;
                           data[i*step + j*nchannels + 1] = 65280;
                           data[i*step + j*nchannels + 2] = 65280;
                           count++;
                    }
             }
      }
      if(count > 40000){
             unrecognized[num] = false;
      }else{
             unrecognized[num] = true;
             cvCircle(tiff, pic_center, cvRound(116 + 50) ,cvScalar(0,0,0), 100);
      }
      if(unrecognized[num] && (num == 0 || num == 10)){
             radius = radius + sqrt(count / M_PI);
      }
      if(unrecognized[num]){
             stat_spot_diameter[num] = sqrt(count / M_PI);
      }
      return tiff;
}
```

```
IplImage* edgeDetector(IplImage *tiff, int num){

      int width     = tiff->width;
      int height    = tiff->height;
      int nchannels = tiff->nChannels;
      int step      = tiff->widthStep/ sizeof(unsigned short);
      unsigned short *data = ( unsigned short* )tiff->imageData;
      int r, g, b;
      CvPoint pic_center = cvPoint(tiff->width / 2, tiff->height / 2);

      if(unrecognized[num]) {
             for(int i = 0; i < height; i++)    {
                    for(int j =0; j < width ; j++)     {
                           CvPoint point = cvPoint(cvRound(j),cvRound(i));
                           b = data[i*step + j*nchannels + 0];
                           g = data[i*step + j*nchannels + 1];
                           r = data[i*step + j*nchannels + 2];
                           if(b == 0 && g == 0 && r == 0){
                                  polarCordinates(1, point.x -  pic_center.x, point.y
- pic_center.y, num);
                           }
                           else{
                                  polarCordinates(2, point.x -  pic_center.x, point.y
- pic_center.y, num);
                           }
                    }
             }
      }
      return tiff;
}
```

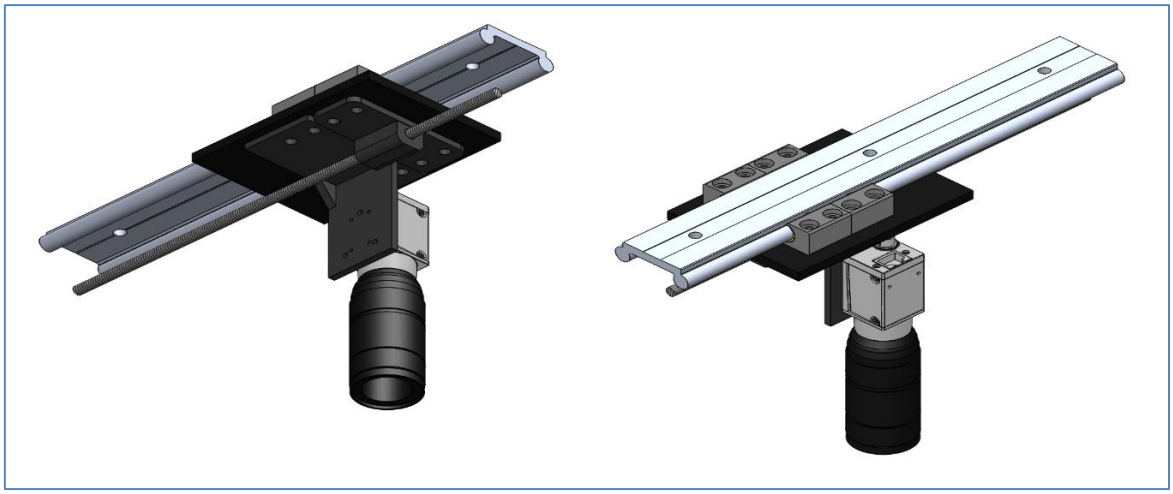
```
IplImage * extractContour(int num){

	if(unrecognized[num]) {
		for(int p =0; p < 360; p++){

			int max = 0;
			for(int i = 0; i < White[num][p].size(); i++){
				if(White[num][p][i] > max){
					max = White[num][p][i];
				}
			}
			contour[num][p] = max;
		}
	}
}
```

Приложение 2

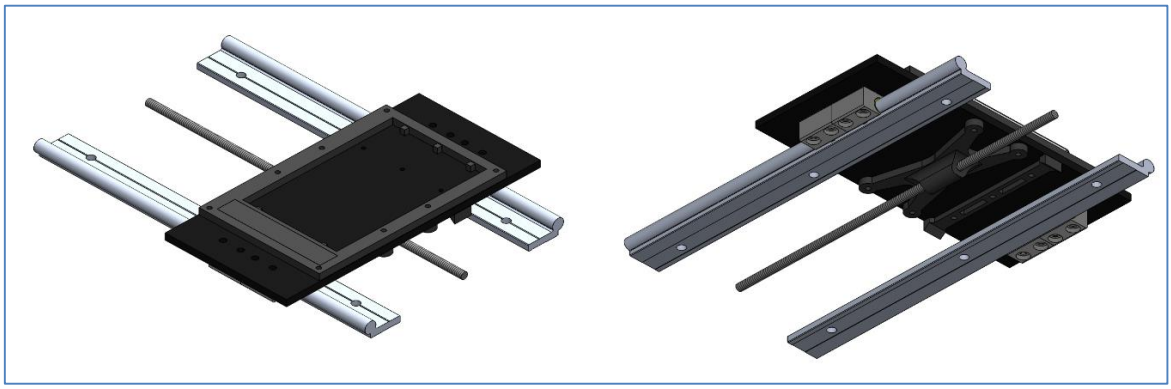


Пример бракованных patch-ей

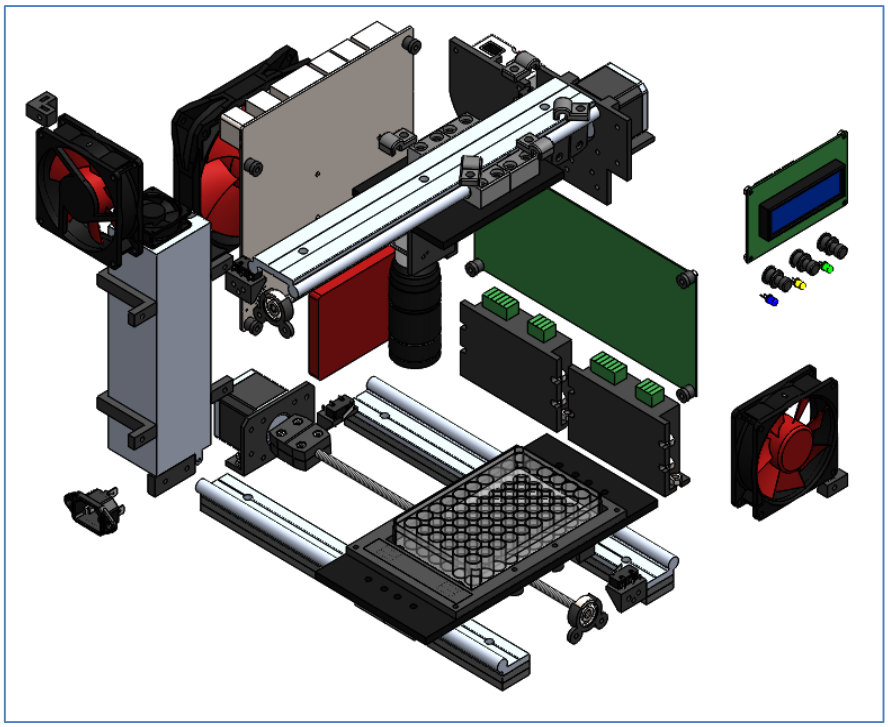
Приложение 3



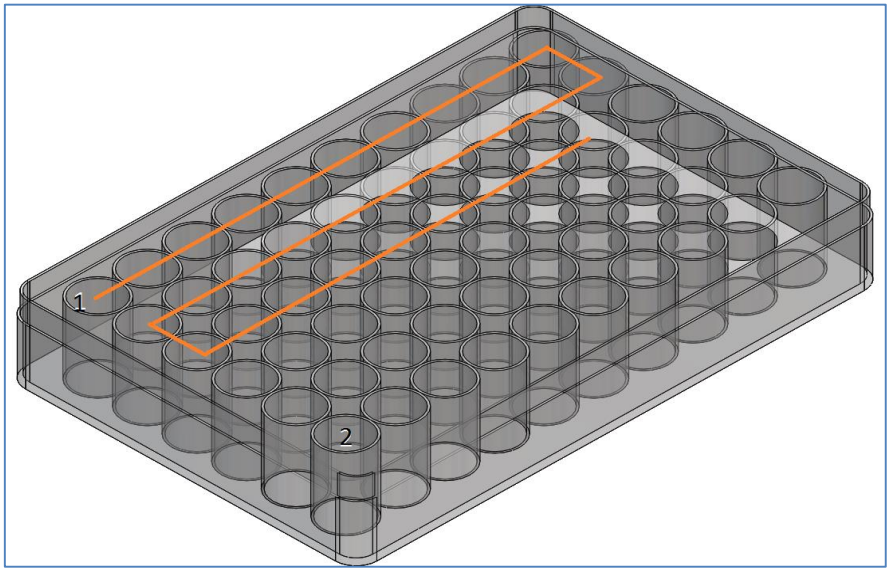
Модуль камеры сканера



Модуль платформы сканера



Компоненты сканера без корпуса



Палетка размещающая patch-и