Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки Программная инженерия

РАБОТА ПРОВЕРЕНА
Рецензент,

_____
«____»_____2017г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н.,
доцент
_____А.А.Замышляева
«____»_____2017 г.

Разработка программного модуля определения глаукомы, используя результаты исследования полей зрения

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–09.03.04.2017.58.ПЗ ВКР

Руководитель работы, ст. преп.
_____/М.Ю.Сартасова
«____»_____2017 г.
Автор работы
Студент группы ЕТ-484
_____/ И.К.Вачков
«____»_____2017 г.
Нормоконтролер, доцент
_____/Т.Ю. Оленчикова
«____»_____2017 г.

Челябинск 2017

АННОТАЦИЯ

Вачков И. К. Разработка программного модуля определения глаукомы, используя результаты исследования полей зрения.– Челябинск: ЮУрГУ, ММиКН-484, 58 с., 24 ил., библиогр. список – 20 наим., 1 прил.

В работе была подробно разобрана предметная область. Были рассмотрены существующие библиотеки компьютерного зрения, в том числе, исследованы методы определения границ объекта на изображении, в частности, зрачка на изображении глаза. Были рассмотрены языки и среды программирования для мобильных устройств.

В качестве метода определения границ объекта был выбран метод «Преобразование Хафа» для определения границ окружности при неизвестных координатах центра.

Были разработаны алгоритмы определения контуров кольца и реализована программа определения глаукомы, используя результаты исследования полей зрения.

Построенную модель можно улучшить для повышения точности определения глаукомы. Также можно изменить метод поиска контуров кольца.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....6
1 ОБЗОР СУЩЕСТВУЮЩИХ СПОСОБОВ И МЕТОДОВ РЕШЕНИЯ ЗАДАЧИ РАСПОЗНАВАНИЯ ГЛАУКОМЫ.....8
1.1 Основные понятия и определения.....8
1.2 Анализ существующих библиотек компьютерного зрения.....11
1.3 Преимущества библиотеки OpenCV.....18
1.4 Метод определения границы зрачка на изображении глаза.....20
1.5 Выбор языка программирования.....24
1.6 Выбор среды разработки.....27
1.7 Выводы по разделу.....28
2 МАТЕМАТИЧЕСКАЯ МОДЕЛЬ.....29
2.1 Преобразование Хафа.....29
2.2 Выводы по разделу.....31
3 РАЗРАБОТКА ПРОГРАММЫ.....32
3.1 Алгоритм определения контуров окружностей.....32
3.2 Алгоритм определения контуров кольца.....33
3.3 Разработка пользовательского интерфейса.....36
3.4 Проверка работы программы на экспериментальных данных.....38
3.5 Выводы по разделу.....42
ЗАКЛЮЧЕНИЕ.....43
БИБЛИОГРАФИЧЕСКИЙ СПИСОК.....44
ПРИЛОЖЕНИЕ 1 ТЕКСТ ПРОГРАММЫ.....46

ВВЕДЕНИЕ

Глаукома – тяжелое заболевание органа зрения, получившее название от зеленоватой окраски, которую приобретает расширенный и неподвижный зрачок в стадии наивысшего развития болезненного процесса – острого приступа глаукомы.

В настоящее время отсутствуют единые представления о причинах возникновения и механизмах развития этой болезни, встречаются определенные сложности даже в самой попытке определить понятие «глаукома».

Сегодня глаукомой принято называть хроническую болезнь глаз, характеризующуюся постоянным или периодическим повышением внутриглазного давления с развитием трофических расстройств в путях оттока внутриглазной жидкости. Таким образом, термин «глаукома» объединяет большую группу заболеваний глаза – около 60.

Глаукома может возникнуть в любом возрасте, начиная с рождения, но распространенность заболевания значительно увеличивается в пожилом, и в старческом возрасте. Так, частота врожденной глаукомы составляет 1 случай на 10-20 тысяч новорожденных, в возрасте 40-45 лет первичная глаукома наблюдается примерно у 0.1% населения. В возрастной группе 50-60 лет глаукома встречается уже в 1.5% случаев, а у лиц старше 75 лет более чем у 3%. Это заболевание занимает одно из первых мест среди причин неизлечимой слепоты и имеет важнейшее социальное значение.

После 40 лет каждому человеку необходимо проходить профилактический осмотр у врача-офтальмолога не реже 1-2 раза в год. Особенно важно это для пациентов с дальнозоркостью, наследственностью по глаукоме и после глазных операций.

К тому же, у использования линейки Поляка для определения внутриглазного давления в методе Маклакова есть несколько недостатков:

1) инструмент довольно быстро загрязняется;
2) для того, чтобы измерить внутриглазное давление по линейке, приходится ждать до 10 минут, пока отпечаток высохнет. При этом, отпечаток может получиться неудачным. Тогда необходимо повторить замеры, и снова ждать, пока отпечаток высохнет;
3) когда порядок внутриглазного давления слишком высок, врачу сложно определить точное значение по линейке;
4) для измерения линейкой необходимо правильно расположить отпечаток и правильно расположить источник света.

Поэтому, в качестве замены линейки Поляка можно разработать программу, которая бы выполняло те же функции. В современном мире телефон с камерой есть почти у каждого, поэтому приложение для телефона будет наилучшим выбором для реализации такой программы: это быстро – просто нужно сделать

фотографию и получить результаты; это удобно – фотографию можно сделать при любом освещении; это точно – исключается человеческий фактор.

Цель работы: разработать программу, которая могла бы по результатам полей зрения определить внутриглазное давление и вероятность глаукомы у пациента. Представить полученные результаты: внутриглазное давление – в виде числа, взяв за основу измерения линейки Поляка; вероятность глаукомы у пациента – в виде процента, исходя из степени отклонения от нормальных данных.

Задачи: для того, чтобы построить систему, необходимо:

1) изучить существующие методы определения границ объекта на изображении, в частности, зрачка на изображении глаза;
2) изучить существующие библиотеки компьютерного зрения;
3) разработать алгоритм интерпретирования полученных данных с изображения.

Предметом разработки является программа, которая определяет внутриглазное давление и вероятность глаукомы у пациента.

Полученные результаты в дальнейшем можно применить для дальнейшей диагностики глаукомы у пациента.

1 ОБЗОР СУЩЕСТВУЮЩИХ СПОСОБОВ И МЕТОДОВ РЕШЕНИЯ ЗАДАЧИ РАСПОЗНАВАНИЯ ГЛАУКОМЫ

1.1 Основные понятия и определения

«Тонометрия глаза – это общее название процедуры по измерению давления внутри глазных яблок. Если говорить точнее, то тонометрия измеряет уровень давления внутриглазной жидкости. Можно сказать, что во время данной процедуры определяется «упругость/наполненность» глазного яблока. В основе измерения лежит степень деформации глазных яблок, когда на роговицу оказывается давление снаружи. То, как деформируется глазное яблоко при подобных манипуляциях, напрямую зависит от внутриглазного давления.»[1]

Измерение и исследование изменения внутриглазного давления требуется для того, чтобы диагностировать у пациента такое опасное заболевание, как глаукома.

Тонометрия может быть проведена различными способами. Результаты процедуры и норма могут отличаться в зависимости от того, какой метод использует офтальмолог. На сегодняшний день наиболее популярны три метода:

1) пальцевый;
2) по Маклакову;
3) бесконтактный.

Рассмотрим метод определения внутриглазного давления по Маклакову.

«Суть метода измерения внутриглазного давления по Маклакову заключается в следующем: грузик давит на глазное яблоко, и чем оно мягче (то есть давление в нем ниже), тем большие площади соприкасаются и, соответственно, больше краски остается на глазах. Данный способ более точен в отличие от пальцевой тонометрии.»[2]

Данный метод имеет ряд ограничений по применению. Например, тонометрия по Маклакову недопустима при воспалительных заболеваниях глаз, а также после операции на глазах. Эти ограничения исходят из процесса проведения процедуры.

Методика Маклакова состоит из следующих шагов:

1) в глаза закапывают специальные капли-анестетик, чтобы пациент не испытывал боли и неприятных ощущений во время процедуры;
2) когда капли подействуют (эффект наступает примерно через 3-5 минут), на глазные яблоки ставят особые грузики (полые металлические цилиндры массой 5, 7,5, 10 и 15 г.), смоченные в красящем растворе. Давление в каждом глазу измеряется по отдельности;
3) затем делают отпечаток глаз на листке бумаг.

Для измерения тонограмм необходимо иметь бинокулярную лупу, настольную лампу, измерительную линейку Б. Л. Поляка, мягкий, остро отточенный карандаш. Изучать стоит только тонограммы с достаточно четкими контурами, в том числе такие, которые имеют ясно видимую овальность. Пользоваться измерительной линейкой можно после полного высыхания отпечатка, то есть через 5-10 минут после тонометрии. Иначе тонометрическая линейка быстро

загрязняется. По этой же причине ни в коем случае нельзя наносить на тонометрический бланк непосредственно перед измерением любые метки чернилами, в особенности, — шариковой ручкой.

Бланк с тонограммами кладется на твердую поверхность (стол, книга). Лампа ставится за тонометрический бланком, но несколько правее или левее его, для того, чтобы избежать рефлексов от поверхностей прозрачной измерительной линейки. Смотреть на отпечатки в процессе измерения нужно через бинокулярную лупу и строго сверху. Поскольку такое положение головы не очень удобно, бинокулярную лупу следует надвигать возможно ниже на переносицу. Для точных измерений линейку нужно переворачивать эмульсионным слоем вниз, чтобы между измерительной сеткой и тонограммой не оставалось зазора даже на толщину пленки, который может вносить параллактические ошибки в результаты исследования. Конечно, считывать цифры внутриглазного давления в зеркально перевернутом виде первое время бывает неудобно, но впоследствии особых трудностей не возникает.

Перед измерением нужно оценить все сделанные тонограммы и отмечать негодные отпечатки, перечеркивая их карандашом. Затем определяют разметку овальности кружков. Для этого вращают тонографический бланк на поверхности стола, чтобы видеть отпечатки с разных сторон. Так намного легче бывает заметить овальность отпечатка и установить направление короткой оси. Это направление, то есть более узкий поперечник отпечатка, обозначается двумя стрелками с обеих сторон каждой тонограммы, но так, чтобы штрихи на сам отпечаток не заходили (рисунок 1.1).

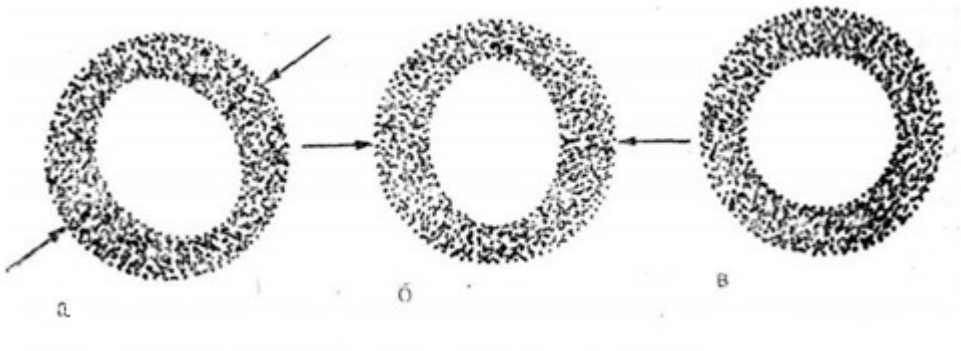


Рисунок 1.1 – Схема разметки тонограмм

Если выделить наименьший диаметр в силу правильности кружка не представляется возможным (схема «в»), отметок не делается. Следует еще раз подчеркнуть, что установить сам факт овальности кружка, если она выражена незначительно, практически невозможно без того, чтобы довольно быстро поворачивать листок с отпечатками перед глазами на 360°, оценивая видимый поперечник кружка всякий раз только в направлении, совпадающем с межзрачковой линией наблюдателя, а не сразу по всем меридианам. Причем, для уверенного выделения минимального поперечника отпечатка бывает необходимо выполнить эту процедуру несколько раз.

Вписывание выделенного поперечника каждого отпечатка в измерительную линейку, соответствующую весу грузика, удобнее вести с более широкого ее конца, который ориентируется «от себя». Желательно передвигать линейку по бланку так, чтобы одна из ограничительных линий все время двигалась по касательной к соответствующему контуру кружка (рисунок 1.2).

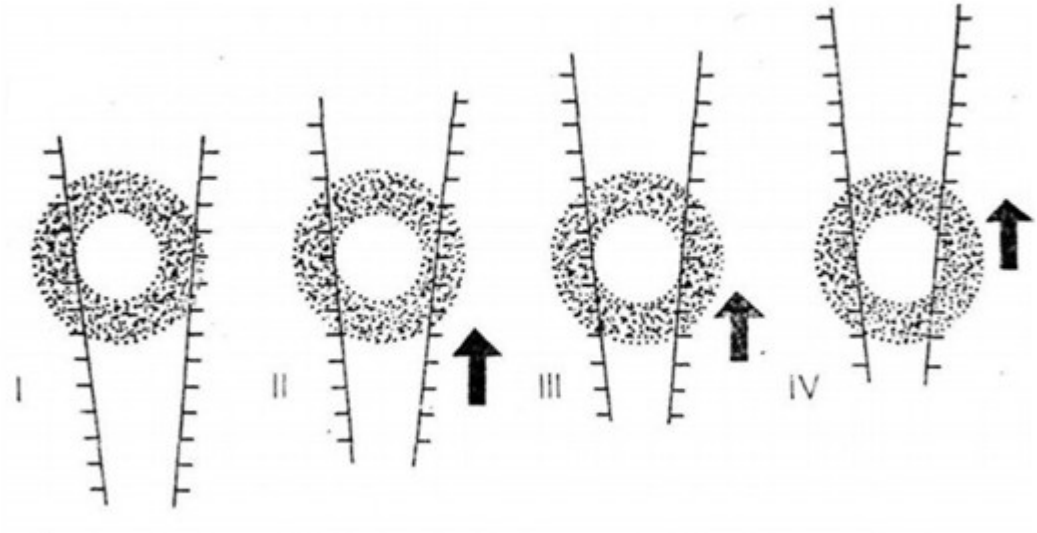


Рисунок 1.2 – Использование линейки для измерения тонограмм

При этом линейка все время плотно прижата к бланку по углам большим и указательным пальцами обеих рук. Движение линейки прекращают, когда создается впечатление, что вторая ограничительная линия вышла при этом в просвет измеряемого кружка (схема IV). Затем возвращают линейку на несколько делений обратно – до момента вписывания кружка в измеритель (схема III). Но остановиться на этом нельзя. Как правило, впечатление «вписывания» создается на уровне не одного какого-то, а двух-трех смежных делений. Если уровень внутриглазного давления невысок, то это не сильно влияет на результат, так как линейка-измеритель покажет одни и те же цифры. Сложнее, когда порядок внутриглазного давления 40-60 мм рт. ст., и цена каждого деления возрастает до ощутимых величин.

В таких случаях советуем прибегать к легким смещениям линейки вправо-влево, буквально на 0,1-0,2 мм, поочередно касаясь контура кружка то левой, то правой ограничительной линией (рисунок 1.3). Если при этом попеременно с обеих сторон отпечатка образуется зазор, – а в динамике его уловить значительно легче, чем при неподвижной линейке, – то значит линейку надо еще поднять вверх (схема I). Если всякий раз линия чуть заползает на кружок, значит линейку нужно опустить (схема III). Обычно замер можно производить без колебаний на уровне, соответствующем среднему из этих двух положений линейки относительно кружка (схема II).

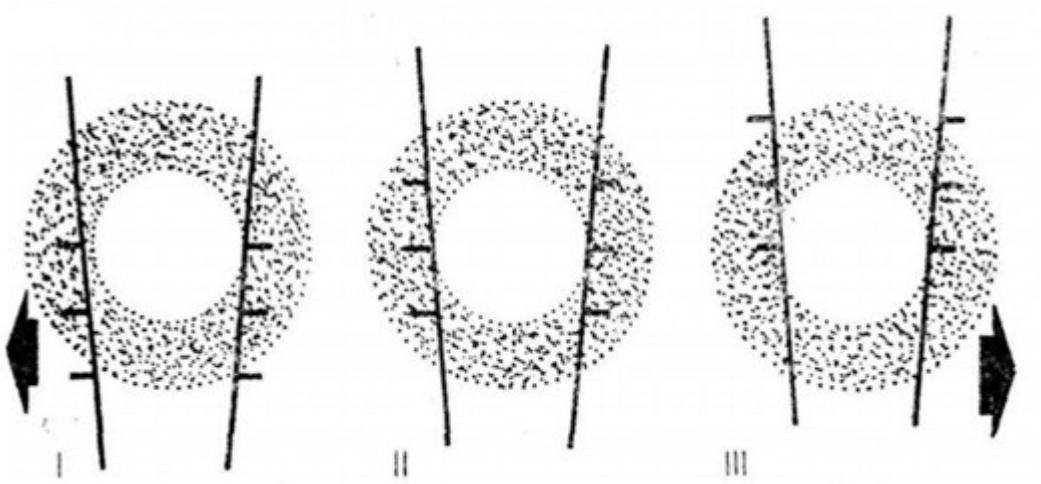


Рисунок 1.3 – Использование линейки для измерения тонограмм при больших значениях внутриглазного давления

Подобный замер каждого кружка полезно выполнить несколько раз, чтобы исключить случайные оценки. Полученные цифры проставляют карандашом на бланке под соответствующими кружками. Если разница между двумя тонограммами, полученными одновременно с одного и того же глаза, не превышает двух миллиметров ртутного столба, окончательная оценка внутриглазного давления выносится по средней между ними величине. Если эта разница превышает 2 мм рт. ст., и оба отпечатка выглядят одинаково качественными, нужно повторить исследование более тщательно.

«Средняя величина тонометрического давления для тонометра Маклакова составляет 19-21 мм рт. ст. Диапазон показателей для здорового глаза варьирует от 18,0 до 25,0 мм рт. ст. Истинное внутриглазное давление всегда ниже тонометрического на 4,5-5,0 мм рт. ст. (измеренного с массой груза 10 г). Средняя величина истинного внутриглазного давления составляет 14-16 мм рт. ст., верхняя граница нормы – 21 мм рт. ст.

Амплитуда колебаний в течение суток может достигать 3-5 мм рт. ст. Как правило, в утренние часы показатели более высокие, чем вечером. При глаукоме давление внутри глазных яблок значительно выше и, соответственно, амплитуда колебаний больше. Поэтому для диагностики глаукомы требуется суточная тонометрия: измерение утром и вечером.»[3]

1.2 Анализ существующих библиотек компьютерного зрения

Компьютерное зрение – это процесс преобразования данных, которые были получены с устройств фото или видео съемки, в новое представление. Данные преобразования выполняются с какой-то определенной целью. В качестве входных данных может быть какая-либо контекстная информация, например, «камера установлена в машине» или «датчик глубины определил объект в радиусе 1 метра». Выходными параметрами для таких входных данных могут быть следующие решения: «есть ли человек в данной сцене» или «есть 14 клеток

опухоли на изображении». Кроме того, новым представлением, полученным из данных с камеры, может быть черно-белое изображение, полученное из цветного или устранение эффекта движения камеры из последовательности кадров.

Задачи компьютерного зрения не являются простыми задачами. Например, процесс поиска на изображении с камеры автомобиля по его образу – довольно сложная задача. Если сравнивать компьютерное зрение с человеческим, то процесс нахождения автомобиля на снимке человеком можно представить как следующую последовательность действий. Для начала человеческий мозг разделяет сигнал, который поступил от зрительного аппарата, на множество каналов. Каждый канал передает информацию различного рода в человеческий мозг. Устройство мозга таково, что он концентрируется на главной информации снимка, при этом остальная информации исключается. На следующем этапе появляется ответный сигнал, перемещающийся в зрительном канале. Кроме того, существуют сигналы, которые поступают на ассоциативные входы. Такие сигналы, которые пришли от датчиков контроля мышц и других чувств, дают возможность мозгу опираться на перекрестные ассоциации, которые были накоплены за все прожитые годы. Из-за наличия обратной связи в мозговом отделе, процесс повторяется снова и снова и включает в себя глаз, который механически управляет освещением с использованием радужной оболочки и настраивает прием на поверхности сетчатки. Из описанного выше видно, что процесс распознавания глазом объекта на снимке является довольно сложным процессом и требует выполнения немалого числа действий человеческим мозгом.

Однако технология компьютерного зрения устроена несколько иначе. Входными данными, которые получает компьютер, является сетка с числами от камеры, либо с диска. У компьютерного зрения нет тех возможностей, которыми управляет человеческий глаз. Отсутствует возможность встроенного распознавания образов, нет автоматического управления фокусом и диафрагмой, нет перекрестных ассоциаций с опытом работы, накопленным за всю человеческую жизнь. Существующие на сегодняшний день системы компьютерного зрения являются довольно простыми.

Кроме того, существует еще одна проблема, связанная с компьютерным зрением. Данная проблема связана с тем, что компьютер воспринимает только двумерные изображения, в свою очередь как каждый предмет обладает трехмерностью. Восприятие трехмерного объекта может радикально меняться в зависимости от позиции, с которой на него смотреть. Данная проблема не имеет однозначного и окончательного решения, имея только двумерное представление трехмерного мира, невозможно однозначно восстановить трехмерный сигнал. 3d сцену можно представить как очень большое количество двумерных изображений. Однако данное представление не является совершенным. Это связано с тем, что данные могут подвергаться различным шумам и искажениям. А также возможны изменения в окружающем пространстве, например, изменения погоды, освещения, отражения, перемещения и многого другого. Также проблемы могут возникнуть из-за каких-либо недостатков в объективе, механических установках, конечном времени интеграции на датчике, которое может привести к

размытию движения, электрическим шумам на датчике или разнообразной другой электронике, артефактам, появившимся после сжатия и захвата изображения. Чтобы объект правильно распознавался, необходимо учитывать все перечисленные выше проблемы.

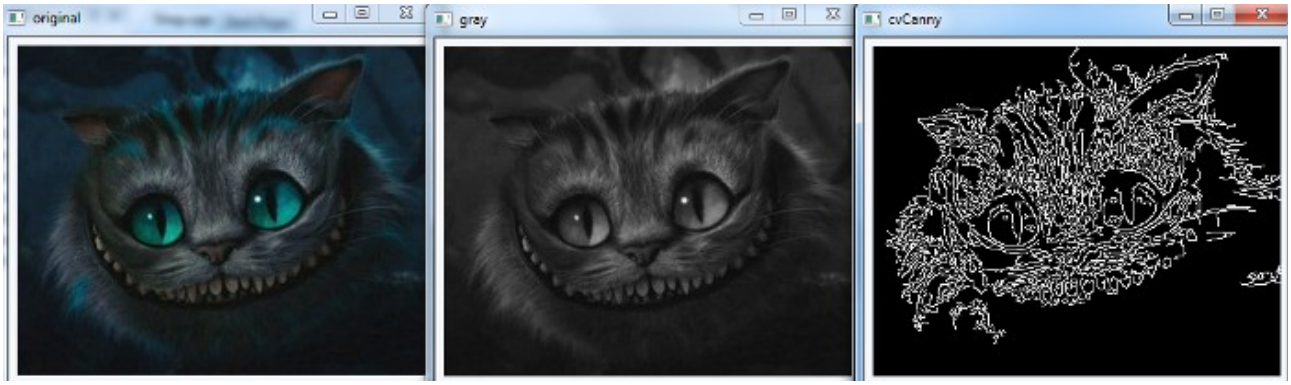


Рисунок 1.4 – Определение контуров объекта на изображении
с использованием детектора границ Кэнни

Очень часто при проектировании практической системы могут учитываться дополнительные знания контекста. Эти знания могут применяться для того, чтобы обойти ограничения, которые были наложены визуальными датчиками. В качестве примера можно рассмотреть следующую ситуацию. Например, есть робот, задачей которого является нахождение ручки на сцене, полученной с камеры. Робот может опираться на следующий факт, стол — это объект, который очень часто располагается в офисах и рабочих пространствах, и именно на столе можно найти ручку. Знание данного факта дает ссылку на размер ручки, она явно должна располагаться на столе, то есть ее размер должен быть много меньше размеров стола. Кроме того, использование данного факта помогает устранить некоторые ложные распознавания ручки в местах, где она не может располагаться, например, на кровати или на стене.

Если, например, за окном есть рекламный постер с изображением ручки, то робот может его проигнорировать, это связано с тем, что на фоне ручки нет древесины стола. Существуют также случаи, когда изображение приходится принимать из базы данных, в которой ручка может оказаться очень больших размеров или, например, быть необычной формы. Данный факт повлечет за собой то, что данные изображения будут исключены из рассмотрения. Также необходимо учитывать, что обычно люди снимают предметы в центре изображения и спереди, например, никто не будет снимать человека вверх ногами. Все это дает неявную информацию по предмету, который необходимо распознать.

Информацию по контексту можно также смоделировать, используя методы машинного обучения. Данный способ является способом явного моделирования. Скрытые переменные, к числу которых можно отнести, размер, ориентацию, положение предмета и многое другое, могут быть соотнесены с их значениями в маркированном обучающем множестве. Также можно попытаться изменить

скрытые переменные смещения с помощью некоторых дополнительных датчиков. С помощью датчика глубины можно довольно точно измерить размер объекта.

Рисунок 1.5 – Результаты работы детектора границ Кэнни
при увеличении размера оператора Собеля

Еще одна проблема компьютерного зрения – это шумы. Очень часто с понятием шумов можно столкнуться, когда используются статистические методы. Часто становится невозможным определение контура путем сравнения соседних точек. Однако если собирать статистику по локальной области, то задача обнаружения контура облегчается. Контур получается в виде строки ответов по локальной области, каждый из которых ориентирован в соответствии со своими соседями. Также существует возможность компенсирования шума путем получения статистических данных в течении довольно долгого периода времени. Однако существуют и другие методы учета шумов и искажений. Данные методы основаны на создании четких моделей, позволяющих распознавать предмет из имеющихся данных. В качестве примера искажений, которые можно так исправить, можно привести искажения от объектива. Для того чтобы полностью избавиться от таких искажений, необходимо только знать параметры простой полиномиальной модели. [4]

Все действия, которые выполняются в компьютерном зрении, происходят из расчета тех данных, которые были получены с камеры в контексте поставленной цели и задачи. Например, можно избавиться от шума и повреждений на изображении таким образом, что система безопасности будет оповещать, если кто-то пересек поставленную границу. Однако программное обеспечение, которое будет использовать технологии компьютерного зрения для роботов, перемещающихся, например, по территории офиса, будет пользоваться несколько иными подходами, чем стационарные камеры. Это связано с тем, что две эти системы имеют различные контексты и конечные цели. Очень часто контекст

задач компьютерного зрения является ограниченным и, учитывая все эти ограничения, поставленную задачу можно во многом упростить, что приведет к тому, что конечное решение будет более правильным.

Библиотека OpenCV направлена как раз на обеспечение работы инструментов для устранения описанных выше проблем компьютерного зрения.

OpenCV (Open Source Computer Vision Library) – это библиотека компьютерного зрения, которая поставляется с открытым исходным программным кодом. Спектр возможностей данной библиотеки очень широк. В ней собрано большое количество алгоритмов для использования технологий компьютерного зрения. После подключения данной библиотеки к своему проекту пользователь получает доступ более чем к 500 функций, предназначенных для решения разнообразных задач. Помимо алгоритмов для работы с технологиями компьютерного зрения, данная библиотека применяется и для обработки изображений, содержит большое число численных алгоритмов и многое другое.

Главной целью библиотеки OpenCV является предоставление легкого в использовании интерфейса, который поможет облегчить использование технологий компьютерного зрения в довольно сложных приложениях. Функции, которые поддерживает библиотека, охватывают разнообразные сферы компьютерного зрения, от медицины, безопасности и до стереозрения и робототехники. Все это благодаря тому, что компьютерное зрение и машинное обучение — два неразрывно связанных понятия. Кроме того, библиотека OpenCV содержит библиотеку MLL, с английского Machine Learning Library. Даная библиотека является библиотекой общего назначения и ориентирована на распознавание статических образов и технологию кластеризации. Данная библиотека является очень эффективной для решения задач компьютерного зрения, которое как раз и является основой OpenCV. Однако для решения конкретных задач машинного обучения данная библиотека не приспособлена и является довольно обобщенной. [5]

Рисунок 1.6 – Структура библиотеки OpenCV

Помимо библиотеки OpenCV, существуют также и другие библиотеки компьютерного зрения. Например, библиотеки Matrox Imaging Library, Camellia

Library, Open eVision, HALCON, VXL, libCVD, IVT, LTI, AForge.NET и некоторые другие.

AForge.NET является библиотекой с открытым исходным кодом, созданной на языке C#, которая предназначена для разработчиков и исследователей в области компьютерного зрения. Кроме того, в библиотеке есть функционал для разработчиков в области искусственного интеллекта. Спектр возможностей библиотеки довольно широк: обработка изображений, нейронные сети, генетические алгоритмы, нечеткая логика, машинное обучение, робототехника и многое другое.

Библиотека включает несколько основных компонентов. AForge.Imaging – библиотека подпрограмм для обработки изображений и фильтров. AForge.Vision – библиотека компьютерного зрения. AForge.Video – набор библиотек для работы с видео информацией. AForge.Neuro – библиотека для выполнения разнообразных действий и операций с нейронными сетями. AForge.Genetic – библиотека подпрограмм для использования генетических алгоритмов для решения различных задач. AForge.Fuzzy – библиотека для работы с нечеткой логикой. AForge.Robotics – библиотека, обеспечивающая поддержку некоторых методов, применяемых в сфере робототехники. AForge.MachineLearning – библиотека для работы с элементами машинного обучения.

AForge.NET постоянно улучшается и прогрессирует. По данной библиотеке есть большое количество примеров, демонстрирующих ее работу, а также актуальная html-документация, которая помогает начинающим разработчикам, которые хотят применять данный фреймворк в своих проектах. Кроме того, как и у библиотеки OpenCV существует сообщество, где можно задавать вопросы и делиться своими наработками по функциям и компонентам AForge.NET. Однако количество участников сообщества по данной библиотеке все-таки меньше, чем в сообществах по библиотеке OpenCV. Данный фреймворк уступает по популярности OpenCV. Еще одним недостатком является тот факт, что вся документация по библиотеке написана только на английском языке, поэтому у некоторых разработчиков могут возникнуть трудности с изучением и работой с данной библиотекой.

VXL, от английского the Vision-something-Libraries, – это набор библиотек, написанных на языке C++, которые предназначены для научных исследований и реализации технологий компьютерного зрения. VXL была написана в ANSI/ISO C++ и предназначена для портативных платформ.

Библиотека состоит из нескольких основных составляющих: VNL (числа) – численные алгоритмы и контейнеры, например, матрицы, векторы, оптимизаторы и т.д., VIL (изображения) – загрузка, сохранение и редактирование изображений во многих наиболее распространенных форматах (также существует возможность работы с очень большими изображениями), VGL (геометрия) – геометрия точек, кривых и других элементарных объектов в одно-, двух- и трехмерном пространствах, VSL (входный и выходной потоки), VBL (основные шаблоны), VUL (утилиты) – разный функционал для независимых платформ.

Кроме основных библиотек входящих в состав VXL, есть также и дополнительные. Дополнительные библиотеки отвечают за такие понятия, как численные алгоритмы, обработка изображений, системы координат, геометрия камеры, стерео, манипуляции с видео потоком, восстановление структуры при движении камеры, графический дизайн, функции отслеживания, топология, классификаторы, 3d визуализация и многое другое.

Особенность библиотеки заключается в том, что каждый ее компонент может использоваться отдельно, не ссылаясь на другие компоненты библиотеки. Таким образом, в приложении можно использовать только те библиотеки, которые действительно необходимы. VXL используется по всему миру. Библиотека применяется в сфере обучения и промышленности, некоторые ведущие мировые эксперты в сфере компьютерного зрения пользуются данной библиотекой. Существует документация по VXL с описанием каждого класса и функций. Однако недостатком является тот факт, что аналогично AForge.NET вся документация написана на английском языке.

LTI или LTI-lib является объектно-ориентированной библиотекой алгоритмов и структур данных, часто применяется при обработке изображений и в сфере компьютерного зрения. LTI-lib была разработана в техническом университете как часть научно-исследовательских проектов в области компьютерного зрения с технологиями робототехники, распознавания объектов, голоса и жестов. Основной целью разработки данной библиотеки является создание объектно-ориентированной библиотеки на языке C++, что во многом упрощало бы использование кода и его обслуживание, но при этом были бы обеспечены быстрые алгоритмы, которые можно было бы использовать в реальных приложениях.

Библиотека была разработана с применением GCC (набор компиляторов, применяемый для разнообразных языков программирования) под Linux и Visual C++ по Windows NT. Многие классы инкапсулируют Windows/Linux функциональность для того, чтобы упростить решение системных или аппаратныхзадач (например, классы для многопоточности и синхронизации, измерения времени и доступ к последовательному порту).

Остальные классы, количество которых превышает 500, выполняют одну из следующих функций. Функции линейной алгебры: предоставление матриц, векторов, решения линейных уравнений, собственные векторы, расчет статистики и многое другое. Функции классификации кластеризации: радиально базисные функции-классификаторы, методы опорных векторов, метод K-средних, нечеткие C-средства, классификация статистических данных. Функции для обработки изображений: разнообразные подходы к сегментации, линейные фильтры, вейвлет-преобразования и многое другое.

Предоставление инструментов для визуализации и рисования. Самое сложное при разработке алгоритмов обработки изображений в C++ это представление временных образов во время отладки. Благодаря объектно-ориентированной архитектуре библиотеки LTI-lib, для предпросмотра необходимо создать объект представления и дать ему образ, который необходимо показать. Кроме

того, если необходимо нарисовать некоторую дополнительную информацию на изображении, например, текст, эллипс, прямоугольник, линию или точку, можно использовать один из объектов рисования. Такой подход позволяет экономить время.

По данной библиотеке также существует документация, которая предоставляется в свободном доступе в Интернете. Документация по библиотеке написана на английском языке. Что касается лицензии, то библиотека LTI, также как и OpenCV, является свободно распространяемым программным обеспечением. Вся пункты и ограничения по лицензии значатся в GNU Lesser General Public License, кроме того, там же можно посмотреть более подробную информацию по лицензии. Существует большое количество проектов, разработанных с применением технологий компьютерного зрения, для которых использовалась именно библиотека LTI.

1.3 Преимущества библиотеки OpenCV

В качестве библиотеки для создания мобильного приложения с использованием технологии распознавания плоских изображений применяется наиболее популярная на сегодняшний день библиотека OpenCV. Такой выбор связан с тем, что по сравнению с некоторыми другими библиотеками компьютерного зрения, данная библиотека обладает некоторыми преимуществами.

Первым и наиболее значительным преимуществом библиотеки является ее производительность. Библиотека OpenCV имеет функцию ручной настройки на использование высокооптимизированного кода IPP. Было проведено сравнение OpenCV и OpenCV с IPP с двумя другими библиотеками компьютерного зрения LTI и VXL (рисунок 1.7). Как видно из диаграммы, по всем четырем критериям эффективности библиотека OpenCV имеет лучшие результаты, OpenCV + IPP превосходит OpenCV без IPP (диаграмма показывает результаты, пропорциональные времени выполнения для каждой из библиотек по четырем критериям эффективности). [6]

В связи с тем, что требовалось использовать компьютерное зрение в режиме реального времени, производительность являлась основной задачей OpenCV. Тот факт, что OpenCV была написана с помощью высокопроизводительного кода на языках C и C++ никак не связано с IPP, который автоматически подключается к OpenCV для улучшения производительности. [7]

Еще одни преимуществом использования библиотеки OpenCV в своем проекте является наличие HTML-документации, которая поставляется вместе с исходным кодом библиотеки. [8] Документация помогает при разработке проекта. Кроме того, существует более новый тип документации по библиотеке OpenCV, а именно, документация в формате Wiki. Документация содержит инструкции по сборке OpenCV, используя среду разработки Eclipse IDE, распознавание лиц с

OpenCV, библиотеку видеонаблюдения, список литературы, совместимые камеры и ссылки на сообщества. [9]

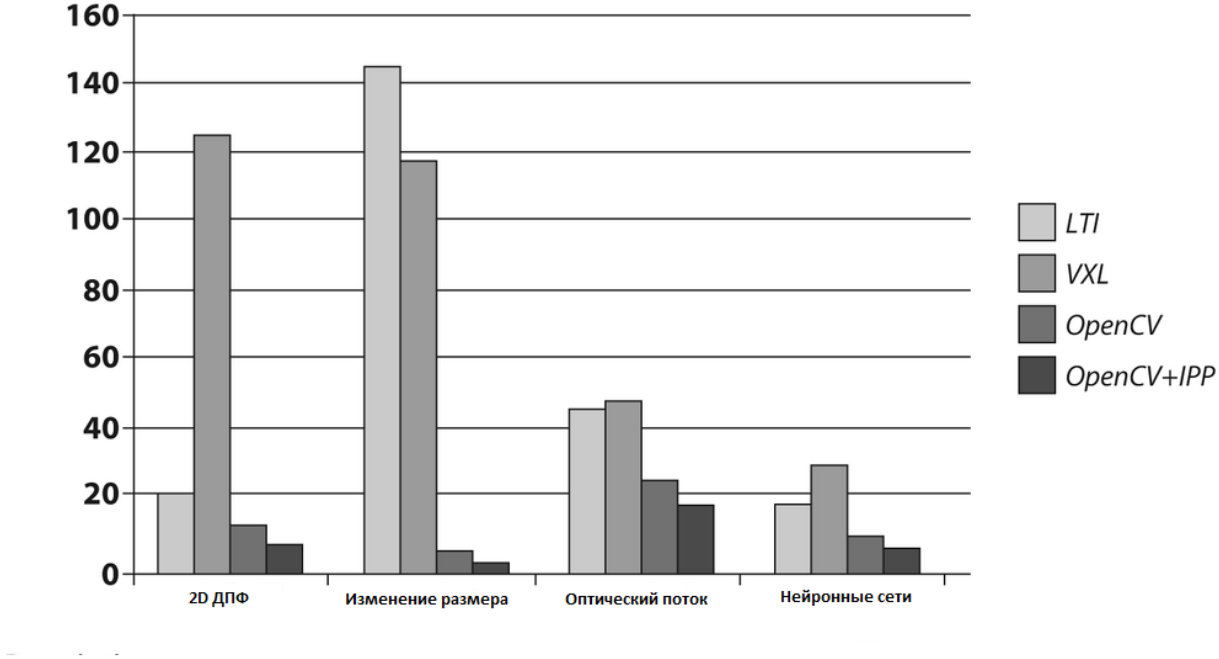


Рисунок 1.7 – Сравнение OpenCV (с и без IPP) и библиотек компьютерного зрения LTI и VXL

Существует также документация в формате Wiki с более уникальной информацией по различным вспомогательным функциям. Данная документация содержит описание следующих понятий: стерео соответствие, точка зрения морфинга камер, 3D-слежение в режиме стерео, объект соответствия, функции для распознавания объектов и внедрение скрытой Марковской модели. [10]

Основная документация по библиотеке OpenCV представлена на английском языке, однако на сегодняшний день существуют и русскоязычные аналоги данной документации, что во многом способствует тому факту, что большинство русскоговорящих разработчиков используют именно OpenCV при создании своих проектов.

Библиотека OpenCV является структурированной библиотекой. Она разделена на пять основных компонентов: алгоритмы обработки изображений, высокоуровневые алгоритмы компьютерного зрения, MLL – библиотека машинного обучения, HighGUI – процедуры и функции ввода и вывода для хранения и загрузки видео и изображений, CXCore – основные структуры данных.

Еще одним преимуществом использования OpenCV является ее переносимость. OpenCV разрабатывалась как портативная библиотека.[11] В самом начале разработка велась на Borland C++, MSVC++ и Intel C++ compiler. Все это означало тот факт, что код на языках C и C++ должен был быть стандартным, чтобы создать облегченную поддержку кроссплатформенности. В начале разработки поддерживаемая архитектура была 32-битной Intel с операционной системой Windows, немного позже с поддержкой ОС Linux. После

того как компания Apple начала использовать процессоры Intel, появилась также поддержка MAC OS X. Немного позже появилась поддержка и 64-битной архитектуры Intel. На сегодняшний день библиотека OpenCV адаптирована почти под все коммерческие системы, кроме того, она работает и на процессорах AMD, в которых также доступна IPP для улучшения производительности (при использовании MMX – мультимедийного расширения). [12]

Наличие большого числа функций (более 500), решающих разнообразные задачи от обработки изображений, компьютерного зрения и до обучения машин, также сыграло роль при выборе в качестве библиотеки компьютерного зрения OpenCV. Кроме того, библиотека имеет открытый исходный код и лицензию, которая позволяет даже с использованием функционала библиотеки выпускать коммерческие продукты.

Еще одним преимуществом является наличие русскоязычной документации по библиотеке. Помимо этого, существует огромное число пособий, научных материалов и книг, в которых описываются функции и методы работы с OpenCV. Также в Интернете можно найти огромное число сообществ, в которых разработчики и пользователи библиотеки делятся опытом и отвечают на вопросы, связанные с библиотекой.

Последним и наиболее важным моментом при выборе библиотеки компьютерного зрения стал тот факт, что библиотека OpenCV имеет реализацию для мобильной платформы Android – OpenCV for Android SDK.[13]

1.4 Метод определения границы зрачка на изображении глаза

Рассмотрим несколько решений задач по определению границ зрачка на изображении глаза.

На изображении глаза с прилегающими частями лица (брови, нос, часть щеки) необходимо найти окружность, являющуюся наилучшей аппроксимацией границы зрачок-радужка. Граница может быть частично закрыта веками, ресницами, бликами. Зрачок является тёмным, но не обязательно самым тёмным объектом на изображении (рисунок 1.8).

Рисунок 1.8 – Исходное изображение

Обозначим исходное растровое изображение I(x,y)=I(p). Необходимо найти координаты и радиус окружности, аппроксимирующей зрачок. На изображении методом бинаризации выделяются компоненты связности C, соответствующие тёмным областям. Для каждой компоненты определяется её граница $L(C)=\{I_i\}$ – циклическая последовательность пикселей. Следует отметить, что количество пикселей в такой последовательности значительно меньше числа пикселей с высоким градиентом яркости, которые можно было бы считать граничными. Зрачок не всегда граничит только с более светлой радужкой, но часто бывает прикрыт веками и/или ресницами, что создаёт помехи двух видов: граница зрачка на изображении присутствует лишь частично; яркость век и ресниц может быть такой же или даже ниже яркости зрачка, и при выделении компоненты связности зрачок сливается с ними (рисунок 1.9).

Рисунок 1.9 – Результат бинаризации

В этом случае лишь небольшая часть границы компоненты связности является границей зрачка, но метод должен определить ту часть границы, которая представлена на изображении. Проводится преобразование Хафа, состоящее в построении в аккумуляторном пространстве луча из каждого пикселя границы перпендикулярно её направлению. В результате, если граница содержала дуги окружности, в аккумуляторе возникают значимые локальные максимумы на месте центров гипотетических окружностей, содержащих эти дуги. Глобальный максимум аккумулятора соответствует наиболее правдоподобному положению окружности. Для определения её радиуса строится гистограмма расстояний от найденного положения центра до граничных пикселей. Максимум гистограммы соответствует радиусу. Из нескольких гипотетических окружностей, построенных таким образом для разных компонент связности, выбирается наилучшая, согласно критериям качества. [14]

Определение порогов бинаризации. Поскольку заранее неизвестны уровни яркости зрачка и радужки, а, следовательно, порог бинаризации, который разделит их, производится несколько проходов (обозначим их количество K)

алгоритма при разных порогах. Величины используемых порогов определяются так, чтобы количество пикселей с яркостью ниже порога составляло определённую долю их общего числа на изображении. Для этого используется гистограмма яркости:

$$H(b)=\left|\left(p:I(p)\le b\right)\right| \quad (1)$$

Операция бинаризации проводится так, чтобы обнулить яркости, превышающие порог, и выделить пиксели с меньшими яркостями, с тем, чтобы далее рассматривать объекты, полученные из тёмных областей.

Выделенные на этапах бинаризации компоненты необходимо обработать преобразованием Хафа. Однако уже по статистическим признакам этих компонент до преобразования можно отбросить их часть, заведомо не содержащую зрачка, а оставшиеся отсортировать так, чтобы обрабатывать вначале компоненты, содержащие зрачок с наибольшей вероятностью. Для удаления заведомо не содержащих зрачок (или имеющих неадекватную форму) компонент связности используются параметры эквивалентного эллипса, имеющего те же моменты, что и компонента. Компонента связности, содержащая зрачок, должна иметь достаточно большую площадь, а отношение длины большой полуоси к малой не должно быть слишком большим. Кроме того, в случае, когда на изображении глаза ресницы/веки имеют яркость такую же или меньшую, чем зрачок, искажение формы компоненты связности, вносимое этим, вытягивает её по горизонтали, но не в вертикальном направлении. Соответственно, объекты с эквивалентным эллипсом, имеющим значительный эксцентриситет и большой осью, расположенной ближе к вертикали, должны быть отброшены.

Преобразование Хафа. Проводится процедура голосования: строятся внутренние нормали в каждом пикселе, на луче нормали выбирается отрезок, ограниченный некоторыми значениями x_{min} и x_{max}, после чего значения элементов аккумуляторного пространства в точках отрезка увеличиваются на единицу.

Вектора нормали строится следующим образом: выбираются ближайшие соседи пикселя в списке, где отрицательный индекс соответствует перемещению вверх по циклическому списку, положительный – перемещению вниз. По координатам выбранных пикселей аппроксимируется касательное направление.

Из двух возможных нормальных направлений выбирается внутренняя нормаль, к направлению обхода. Отрезок голосования строится в растре аккумулятора алгоритмом Брезенхема. По окончании процедуры голосования значения в аккумуляторном пространстве сглаживаются фильтром низкой частоты. Центр гипотетической окружности находится как глобальный максимум в сглаженном аккумуляторе.

Как видно, в решении данной задачи по определению контура зрачка на изображении используется преобразование Хафа. Рассмотрим еще один метод определения границ зрачка.

Задача поиска границы зрачка на изображении человеческого глаза является одной из подзадач выделения радужной оболочки глаза в системах биометрической идентификации. Для её решения предложено большое

количество подходов. Одним из них является класс методов, использующих преобразование Хафа. [15]

Методы, использующие преобразование Хафа, являются эффективными на широком классе изображений глаза, однако поиск может занять продолжительное время. С помощью данного метода определяют координаты центра и радиус зрачка глаза.

Для поиска округлой формы зрачка используется преобразование Хафа следующего вида:

$$H(x_0, y_0, r) = \sum_i h(x_i, y_i, x_0, y_0, r) \quad , \tag{2}$$

где

$$h(x_i, y_i, x_0, y_0, r) \begin{cases} 1, если (x_i - x_0)^2 + (y_i - y_0)^2 = r^2 \\ 0, иначе \end{cases} \quad , \tag{3}$$

(x,y)– определяемые координаты центра зрачка; (x_i,y_i)– пиксель на изображении из некоторой окрестности.

На этой основе строится аккумуляторное пространство, которое определяет искомый объект. Для окружности произвольного радиуса аккумулятором является трёхмерное пространство, заданное координатами центра и величиной радиуса. Затем осуществляется процедура голосования: каждой точке изображения ставится в соответствии некоторое множество точек из трёхмерного аккумуляторного пространства. Результатом работы метода является точка аккумуляторного пространства, попавшая в наибольшее число множеств.

Для решения задачи разработан алгоритм, позволяющий локализовать зрачок на изображении глаза за приемлемое время.

Входными данными для задачи является изображение размером WxH пикселей. Алгоритм состоит из следующих шагов:

1. К исходному изображению применяется фильтр Гаусса для фильтрации шумов.
2. Изображение разбивается на прямоугольники, в каждом из которых определяется средняя яркость. Считается, что прямоугольник с минимальной яркостью находится в области зрачка.
3. Изображение бинаризуется с некоторым порогом, после чего остаются компоненты, на которых присутствует зрачок.
4. Определяется область интереса для поиска зрачка.
5. С учётом области интереса применяется преобразование Хафа.

Для сокращения пространства поиска определим приблизительное положение зрачка на изображении, исходя из предположения, что это область наименьшей средней яркости. Всё изображение разбивается на N прямоугольников по горизонтали, для каждого из которых рассчитывается средняя яркость.

Согласно принятому предположению прямоугольник с минимальной яркостью находится в области зрачка, таким образом, определяется приблизительный центр зрачка (рисунок 1.10).

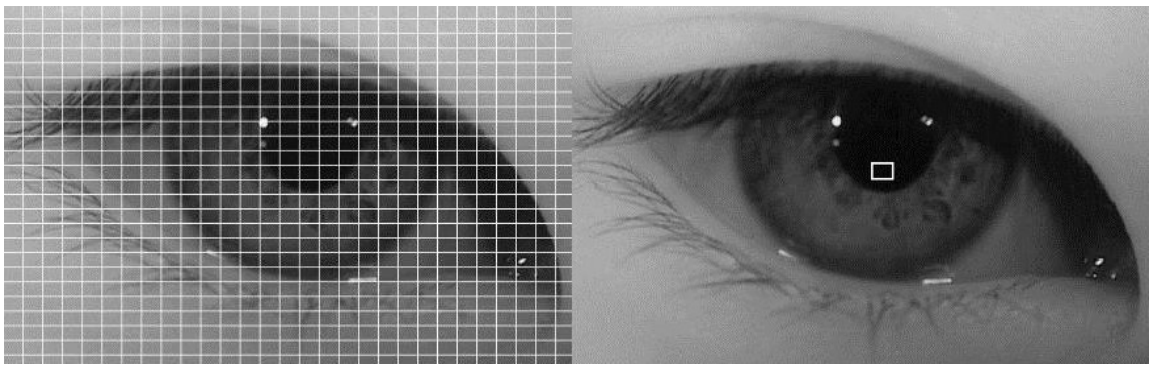


Рисунок 1.10 – Поиск области минимальной яркости

В отличие от проекций яркости данный метод позволяет точнее определить положение зрачка, так как суммируется яркость изображения локально, а не по строкам (столбцам).

Бинаризация представляет собой разделение всех пикселей изображения на два класса по определённому порогу яркости: пикселям, имеющим яркость меньше пороговой, присваивается нулевое значение; пикселям, имеющим яркость больше – присваивается единичное значение. К изображению можно применить сглаживающий фильтр для уменьшения шумов и получения более ровных границ.

Для выявления округлой части границы зрачка используется разновидность преобразования Хафа. Исходя из размеров области интереса, определяются минимальные/максимальные значения центра и радиуса зрачка. Для каждой комбинации таких значений происходит вычисление координат точек окружности и определение значения пикселя на изображении границы зрачка. Если в вычисленной точке значение большее нуля, то аккумулятор суммируется. [16]

Путём голосования выбирается центр и радиус зрачка. Иногда получается так, что зрачок включает в себя часть верхнего века. Такое происходит, когда глаз частично закрыт из-за физиологических особенностей человека или в момент моргания. Такие случаи обрабатываются на следующих шагах в биометрической системы идентификации, когда происходит определение возможной области наложения на радужную оболочку глаза верхнего и/или нижнего век и исключение попавших в эту область пикселей изображения из обработки.

В данном случае также используется преобразование Хафа

1.5 Выбор языка программирования

Одним из самых важных элементов в процессе разработки приложения является выбор правильного языка программирования.

При всем нескончаемом скепсисе, направленным в сторону продуктов Microsoft, стоит признать, что C# этого не заслуживает. Это прекрасный язык, вобравший в себя всё лучшее от Java, при этом учтя и исправив многие недостатки. Что касается разработки приложений под Android, то здесь существуют одни из самых функциональных сред Visual и Xamarin Studio. А еще

знание C# станет приятным бонусом, когда доберетесь до использования Unity 3D. С таким набором возможности будут безграничны.

Тот факт, что Android не поддерживает использование Python для создания нативных приложений, еще не означает, что это невозможно. Было разработано множество инструментов, позволяющих скомпилировать код на Python в требуемое состояние. Самым популярным фреймворком является Kivy, который без труда позволит создать приложение для Play Market на чистом Python.

Официально выпущенный лишь год назад, Kotlin очень быстро завоевывает сердца разработчиков по всему миру практически полным отсутствием недостатков. С его помощью – точнее с помощью родной среды IntelliJ IDEA – не будет чувствоваться никаких проблем в разработке нативных приложений для Android.

Стандартный языковой набор веб-разработчика: HTML, CSS и JavaScript. Отсутствие знаний в этих 3 языках приведет к тому, что разработанные приложения будут достаточно узкой направленности. Даже если вы непосредственно веб-технологий в будущей работе касаться не хотите, то гибридных приложений избежать получиться вряд ли. Работать с HTML, CSS и JavaScript можно используя среды PhoneGap Build или, в более специализированном случае, Adobe Cordova. Больших знаний они не потребуют, а результат обеспечат. Или из последних разработок, React Native от Facebook — это уже следующий уровень удобства взаимодействия, но опыта и документации скопилось мало.

Lua — язык, который старше Java, куда менее популярный, но всё равно востребованный. У него есть ряд преимуществ: динамическая типизация, относительно простой синтаксис. Но до наших дней он дожил благодаря задействованности в играх. Именно удобство создания программной прослойки между движком и оболочкой открыло перед Lua двери в мир карманных гаджетов.

Corona SDK — среда для разработки мобильных кроссплатформенных приложений, преимущественно игр, где главным инструментом является Lua. С 2015 года она распространяется бесплатно, рассчитана на начинающих разработчиков, плюс ко всему вы можете найти много полезной информации, как в англоязычном, так и русскоязычном сегменте интернета.

Java – основной язык для программистов под Android. Основной исходный код Android написан именно на этом языке, именно поэтому большинство выбирают именно этот язык. Приложения, написанные на Java, запускаются в Android с помощью виртуальной машины ART – аналога виртуальной машины Java. Данный язык подходит для широкого спектра задач. [17]

Google на данный момент официально поддерживает достаточно хорошую среду разработки Android Studio, которая собрана на основе Intellij IDEA от JetBrains. Также существует очень подробная документация от Google.

Не будет большим преувеличением назвать Java официальным языком Android. Во всяком случае, почти вся образовательная документация, все интернет-курсы основаны на этом. А еще это самый популярный язык по оценке

TIOBE, второй по количеству исходников на GitHub, да и вообще большой красивый язык. Именно поэтому изучение Java должно быть первоочередной задачей для любого Android-разработчика. Пусть это будет непросто – все-таки языку 22 года, а легкость никогда не была его коньком – пусть теоретически можно обойтись более современными языками, помните – невозможно добиться существенных успехов на Android, абсолютно не понимая Java, не говоря уже о конкретных исходниках.

Кроме того, в качестве среды разработки можно выбрать Eclipse – очень популярную среду для Java-программистов. С официальным плагином ADT от Google данная среда разработки могла бы составить хорошую конкуренцию. [18] Но компания Mountain-View два года назад перестала поддерживать Eclipse, уступив место новейшей Android Studio.

На самом деле Google предоставляет разработчиков две среды разработки: SDK, предназначенная для работы с Java, и NDK, где нативными языками являются C/C++. Конечно не получится написать целое приложение с использованием лишь этих языков, но с их помощью можно создать библиотеку, которую впоследствии при помощи Java подключить к основному телу программы. Несмотря на то, что подавляющему большинству разработчиков нет никакого дела до NDK, тем не менее, задействовав этот инструмент, будут получены лучшие результаты по производительности и использованию внутренних ресурсов. А это именно то, что на Android отличает хорошую идею приложения от хорошей реализации.

Для написания некоторых программ и участков кода, выполнение которых требует максимальной скорости, могут быть использованы библиотеки C/C++. Использование этих языков программирования возможно через специальный пакет для разработчиков Android Native Development Kit, ориентированный специально для создания приложений с использованием C++.

Android разработка на C или C++ может показаться более простой чем на Java, но, несмотря на то, что язык предлагает полную свободу производимых действий, он имеет некоторые специфические особенности, на изучение которых уйдёт много времени. [19]

За всё существование Android было создано немало фреймворков и средств разработки для C++. Особенно выделяется широко известная среда IDE QtCreator, позволяющие разрабатывать кроссплатформенные приложения. [20] Также в качестве среды программирования можно выбрать Android Studio или Visual Studio 2015.

В качестве среды разработки была выбрана Android Studio, поскольку она является универсальной: можно использовать как C++, так и Java. К тому же, у данной среды есть подробная документация от Google.

Язык Java был выбран в качестве основного языка программирования. Такой выбор связан, в первую очередь, с тем, что существует много литературы, в которой описывается работа с OpenCV. К тому же, программирование под Android на Java проще, чем на C++. Но в проекте также используется язык C++, так как все библиотеки, которые используются, написаны именно на этом языке.

1.6 Выбор среды разработки

Следом за выбором языка программирования, необходимо выбрать подходящую среду разработки. Это является одним из самых важных элементов в процессе разработки приложения. Официальными представителями лидеров мобильного рынка являются Windows, Google и Apple.

Visual Studio 2015 является одним из старейших программных продуктов для создания как консольных приложений, так и обладающие графическим интерфейсом. Добавление сторонних плагинов позволяет серьёзно расширить функциональность среды, в том числе до кроссплатформенного состояния. Но у среды есть и минусы: новичку будет просто невозможно самостоятельно разобраться с Visual Studio без прохождения специальных курсов и чтения литературы. Это продукт скорее для опытных разработчиков, обращающих внимание на качество редактора и функции тестирования.

Android Studio является относительно молодая и стремительно развивающаяся IDE, ориентированная на разработчиков приложений для Android. Но у нее в редакторе кода проявляются скупые возможности персонализации.

XCode является IDE, ориентированная на создание приложений для OS X и iOS. Для использования языков Objective C и Swift на сегодня это лучшее, а для некоторых задач и вовсе единственное решение. Но многие разработчики жалуются на стабильность среды, вынуждающую вносить дополнительные изменения в свои проекты после выхода очередной версии. Кроме того, XCode относительно сложная IDE для самопознания новичком.

От официальных представителей перейдём к универсальным кроссплатформенным средам разработки: Xamarin Studio, IntelliJ IDEA, Appcelerator Titanium, Eclipse, Netbeans, PhoneGap.

Xamarin Studio является популярным инструментом разработки приложений под Windows, Phone, Android и iOS, использующий по сути только один язык — C#. Помимо непосредственно Xamarin Studio вы также можете пользоваться плагином для Visual Studio. Но существуют незначительные, но тем не менее регулярные ошибки, как непосредственно в самой IDE, так и в выходном коде. Также, несмотря на репутацию кроссплатформенной среды, портировать уже готовые приложения на Xamarin достаточно затруднительно.

IntelliJ IDEA является IDE, разработанная компанией JetBrains, позволяющая создавать программы на множестве популярных языков, среди которых Java, JavaScript, Python, Ruby, Groovy, Scala, PHP, C, C++. Но главной проблемой среды считается производительность. Томительное ожидание выполнения компиляции, перекомпиляции, тестирования порой действительно раздражает.

Appcelerator Titanium – это платформа для быстрого создания консольных и графических приложений для всех подручных устройств. Возможности, предоставляемые Appcelerator Titanium, имеют и обратную сторону: генерируемые ошибки в коде, искусственные ограничения, недостаточно качественная документация.

Eclipse – это среда разработки, изначально ориентированная на работу с Java, прославилась большим количеством внешних модулей, существенно расширяющих её функциональность (в том числе, это касается количества поддерживаемых языков). Но существенная нехватка документации, отсутствие единого сообщества разработчиков заставляет задуматься при выборе данной среды.

Netbeans является мощной IDE для разработки приложений на Java, JavaScript, Python, PHP, C, C++ и даже Ада. Но среда имеет невысокое быстродействие из-за концепции «всё в одном». Некоторые плагины (в том числе для разработки приложений для Android) имеют существенные ограничения функциональности.

PhoneGap – это необычная среда разработки кроссплатформенных приложений, не требующая знания «родных» языков. То есть для того, чтобы создать приложение для Android, знание Java вам не потребуется. Используются JavaScript в связке с HTML5 и CSS3. Однако у среды ограниченная функциональность, вызванная непосредственно основной идеей нецелевой среды разработки.

1.7 Выводы по разделу

В рассмотренных решениях задачи определения границы зрачка на изображении используется преобразование Хафа. Этот метод довольно популярен и показывает хорошие результаты. Кроме того, реализация данного метода уже существует в библиотеке OpenCV. Поэтому, для решения задачи определения глаукомы, используя результаты исследования полей зрения, также был выбран данный метод.

В качестве среды разработки была выбрана система Android Studio, поскольку она является универсальной: можно использовать как C++, так и Java. К тому же, у данной среды есть подробная документация от Google.

Язык Java был выбран в качестве основного языка программирования. Такой выбор связан, в первую очередь, с тем, что существует много литературы, в которой описывается работа с OpenCV. К тому же, программирование под Android на Java проще, чем на C++. Но в проекте также используется язык C++, так как все библиотеки, которые используются, написаны именно на этом языке.

2 МАТЕМАТИЧЕСКАЯ МОДЕЛЬ

2.1 Преобразование Хафа

Преобразование Хафа – алгоритм, применяемый для извлечения элементов из изображения. Используется в анализе изображений, цифровой обработке изображений и компьютерном зрении. Он предназначен для поиска объектов, принадлежащих определённому классу фигур, с использованием процедуры голосования.

Для фигуры F из определенного класса задается её параметрическое представление Φ вектором параметров t: $F \underset{\leftrightarrow}{\Phi} t$. Тогда параметрическим представлением окружности:

$$\Phi = \left\{ (x - x_0)^2 + (y - y_0)^2 = r^2 \right\}, \tag{4}$$

будет вектор параметров: $t = (x_0, y_0, r)$.

На изображении фигура отображается во множество пикселей:

$$F \underset{\leftrightarrow}{\Phi} \{ p_i \}, \tag{5}$$

где ↔ отношение «один ко многим». В дальнейшем этим знаком мы и будем обозначать такое отношение.

Поиск на изображении всех фигур данного класса производится следующим образом: выбираются пиксели $\{q_j\}$ изображения, которые могут принадлежать таким фигурам – это пиксели с высоким градиентом яркости. Такие пиксели называются кандидатами.

Каждый пиксель-кандидат q может принадлежать одной или нескольким гипотетическим фигурам F_k заданного класса:

$$q \in F_k, F_k \underset{\leftrightarrow}{\Phi} \{ x_{i_k} \} \exists i_k : x_{i_k} = q. \tag{6}$$

Можно записать: $q \underset{\leftrightarrow}{\Phi^{-1}} \{ F_k \}$, учитывая соответствие фигуры и её параметров $q \underset{\leftrightarrow}{\Phi^{-1}} \{ p_k \}$. Таким образом, каждый пиксель-кандидат порождает набор точек в пространстве параметров, что можно представить в виде индикаторной функции:

$$v(q, p) = \begin{cases} 1, \text{если } p \in \{ p_k \} \underset{\leftrightarrow}{\Phi^{-1}} q \\ 0, \text{иначе}. \end{cases}, \tag{7}$$

Пространство p называется аккумулятором, а построение функции $v(q, p)$ - голосованием. Преобразование Хафа заключается в том, что для каждого кандидата помечаются соответствующие точки аккумулятора, то есть суммируются функции-индикаторы:

$$V(p) = \sum_{q_j} v(q, p). \tag{8}$$

Точка аккумулятора, получившая наибольшее количество голосов, то есть максимум $p^{\wedge} = arg\,max_{p} V(p)$, отвечает наилучшему найденному положению фигуры заданного класса на изображении. В случае поиска нескольких фигур рассматривается не единственный глобальный максимум аккумулятора, а несколько локальных. Перед поиском максимумов аккумулятор обычно обрабатывается фильтром низкой частоты.

Пошагово опишем алгоритм обнаружения окружностей заданного радиуса на полутоновых изображениях, использующий оценку ориентации нормали в голосующих контурных точках.

Первым шагом процесса является обнаружение пикселов края, окружающих периметр объекта. Для этого используется оператор Собела, дающий оценку амплитуды и направления вектора-градиента. Голосующими контурными точками считаются точки с высоким значением модуля градиента. Для каждого обнаруженного краевого пиксела используется оценка положения и ориентации контура с целью оценки центра кругового объекта радиуса R путем движения на расстояние R от краевого пиксела в направлении нормали к контуру (то есть в направлении вектора-градиента). Если эту операцию повторять для каждого краевого пиксела, будет найдено множество положений предполагаемых точек центра, которое усредняется для определения точного местонахождения центра.

Но в данном случае радиус окружности является неизвестным, поэтому необходимо включить R в качестве дополнительной переменной в параметрическое пространство-аккумулятор: тогда процедура поиска пика должна определить радиус, так же как и положение центра путем рассмотрения изменений вдоль третьего измерения параметрического пространства.

Пусть для каждого возможного направления на "центр" контурная точка голосует не точкой на расстоянии R, а лучом в этом направлении (рисунок 2.1, а). Таким образом, будут задействованы все возможные положения "центра" при любом масштабе объекта, и это позволит искать окружности независимо от их радиуса (рисунок 2.1, б).

На втором этапе анализа, после обнаружения потенциальных центров окружностей, следует повторно обратиться к изображению и уточнить радиус окружностей с центрами в найденных точках. Заметим также, что направление градиента в точке контура контрастной окружности на изображении есть не что иное, как предел срединного перпендикуляра к секущей окружности при стремлении длины секущей к нулю.

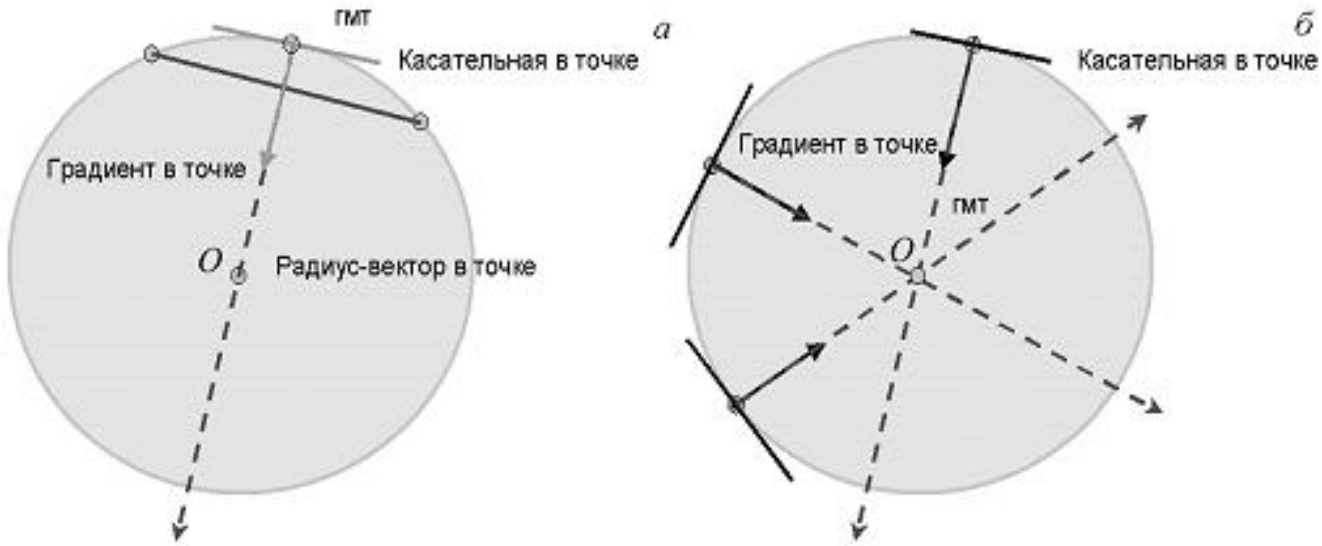


Рисунок 2.1– Принцип обнаружения окружности неизвестного радиуса на полутоновом изображении методом голосования.

Непосредственный поиск трех параметров (координат центра и радиуса окружности) голосованием в трехмерном аккумуляторе:

$$v(q,p)=\begin{cases}1, если \left\|p'-q\right\|=p_3^2,\\ 0, иначе,\end{cases} \tag{9}$$

где p' – двумерный вектор, составленный из первых двух координат трехмерного вектора p, соответствующих координатам центра окружности, p_3 – третья координата этого вектора, соответствующая радиусу. Фактически, в трехмерном пространстве p для каждого фиксированного значения p_3 в плоскости (p_1,p_2) проводится окружность, индикаторная функция $v(q,p)$ представляет собой конус в пространстве параметров с вершиной в точке $(q_1,q_2,0)$ и осью в направлении p_3.

2.2 Выводы по разделу

В результате был определен метод, метод Хафа, алгоритм, применяемый для извлечения элементов из изображения, который позволит найти контуры окружностей. Данный метод уже зарекомендовал себя в решении аналогичных задач, поэтому без сомнений можно использовать метод и для решения текущей задачи. Основным преимуществом данного метода является то, что он позволяет найти все параметры окружности – координаты центра и радиуса. Немаловажно также и то, что уже существует реализация данного метода для мобильных приложений в библиотеке OpenCV.

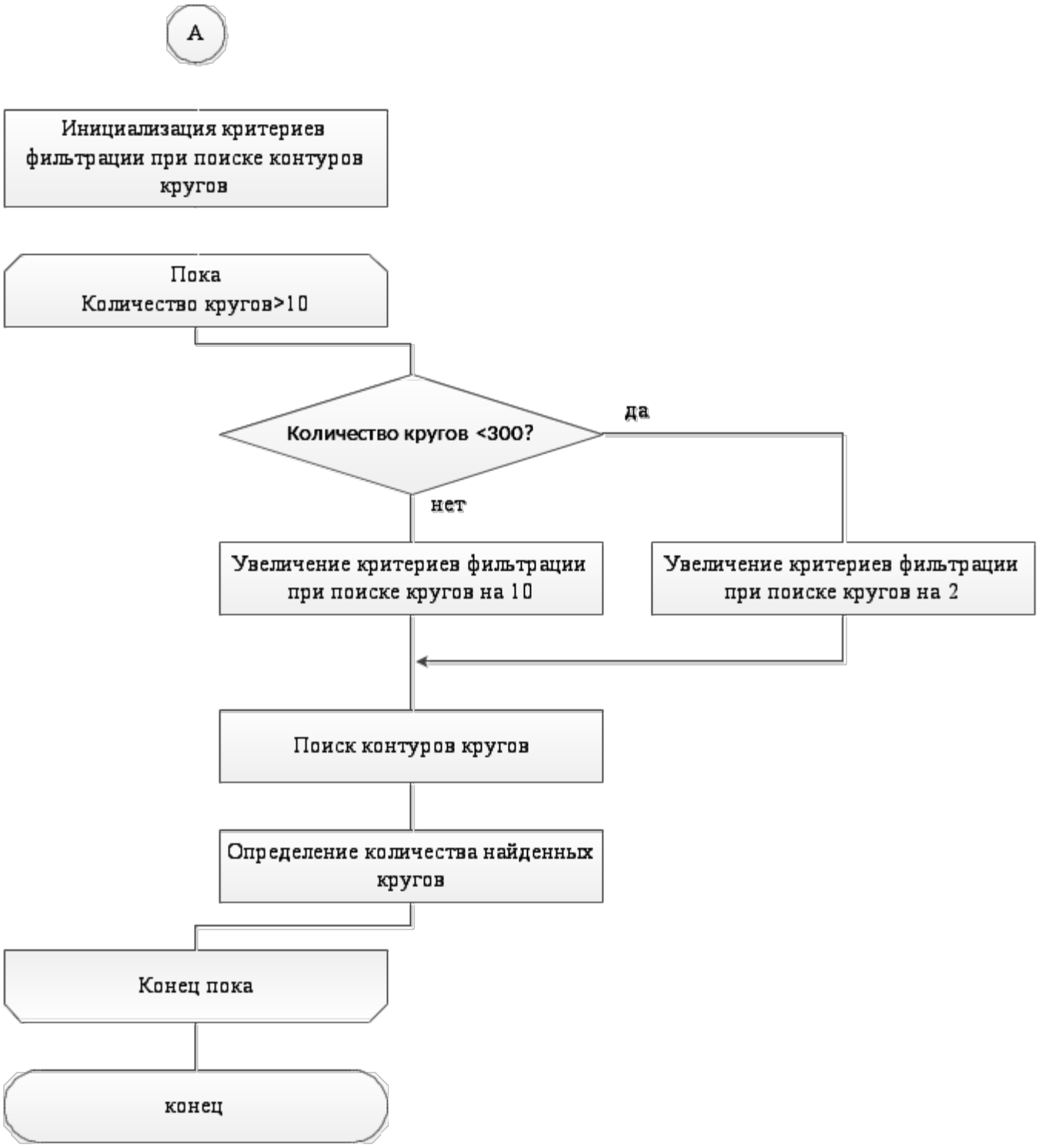
3 РАЗРАБОТКА ПРОГРАММЫ

В данной главе будут рассмотрены алгоритмы, используемые в программе.

3.1 Алгоритм определения контуров окружностей

После того, как пользователь сделал фотографию, необходимо найти контуры кругов. При этом в зависимости от качества фотографии, размера объектов и освещенности, программа распознает или очень много кругов, или не находит ни одного. Поэтому, для решения данной проблемы необходимо задать низкий критерий фильтрации кругов. Затем, постепенно увеличивается этот критерий, пока не останется несколько контуров, из которых максимальный и минимальный и будут искомые контуры. Если же кругов у нас находится слишком много, то для ускорения работы шаг увеличения критерия фильтрации становится больше. На рисунке 3.1 приведена схема полученного алгоритма.

Рисунок 3.1 - Схема алгоритма подпрограммы определения контуров окружностей



Окончание рисунка 3.1

3.2 Алгоритм определения контуров кольца

После того, как были найдены контуры кругов и большинство «ложных» кругов было отфильтровано, все равно останется больше двух кругов. Поэтому, необходимо выбрать из оставшихся контуров кругов те, которые и описывают наше кольцо. Это будут круги с наименьшим и наибольшим радиусом. На рисунке 3.2 приведена схема полученного алгоритма.

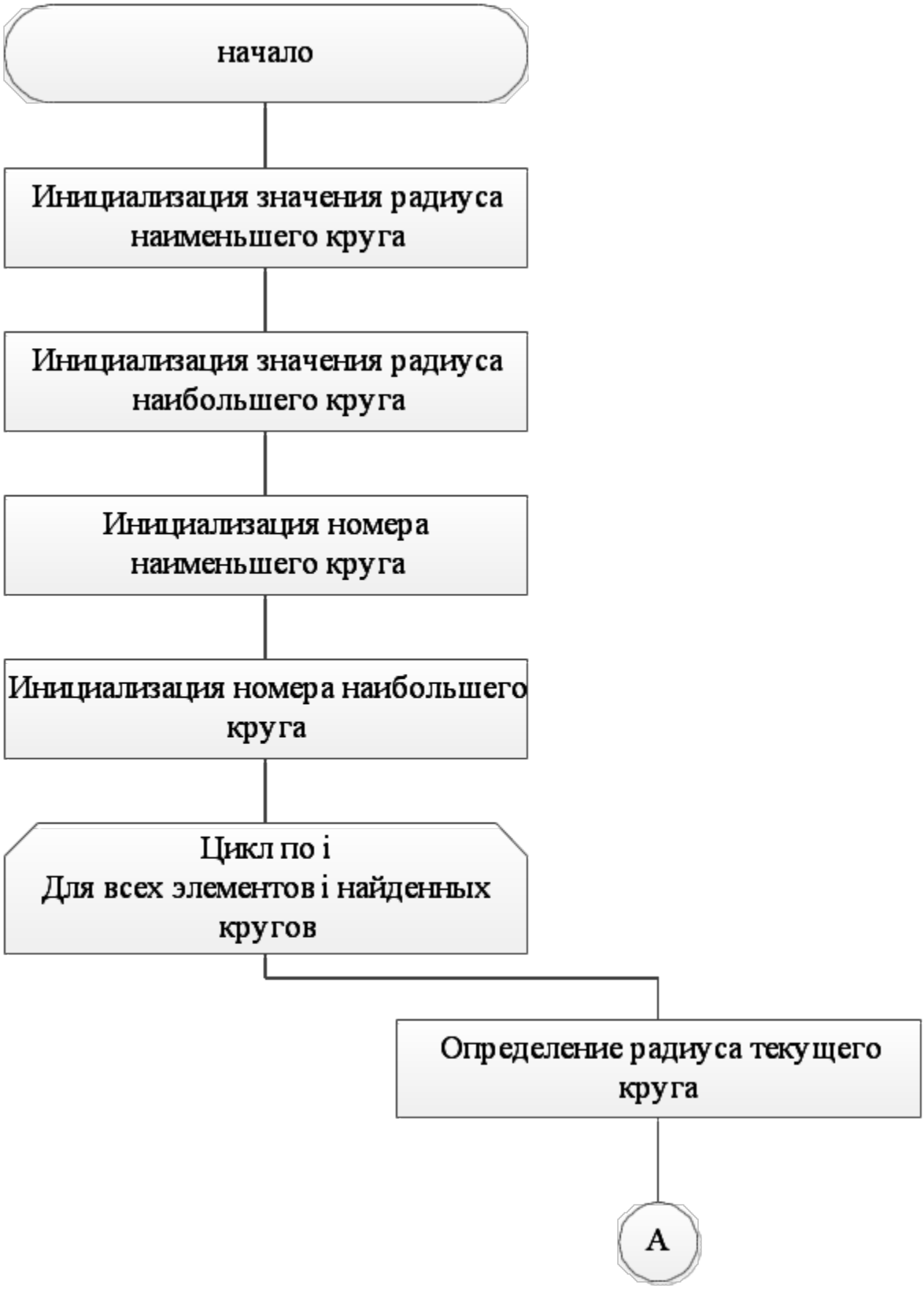
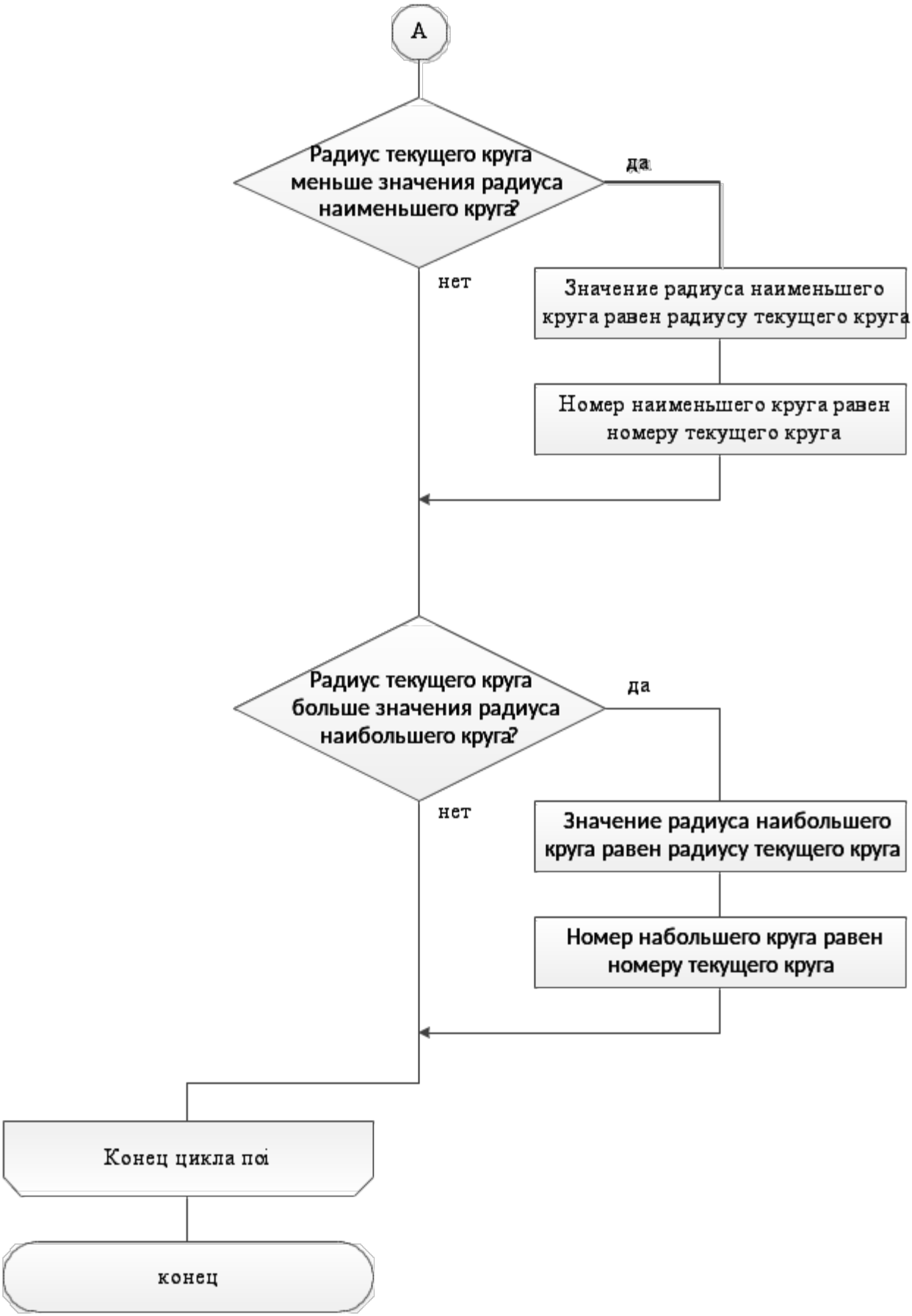


Рисунок 3.2 - Схема алгоритма подпрограммы определения контуров кольца



Окончание рисунка 3.2

3.3 Разработка пользовательского интерфейса

Так как функционал разрабатываемого приложения сильно ограничен решением одной задачи, интерфейс приложения довольно лаконичен.

От пользователя требуется выполнять 2 задачи:

1) сделать фотографию. При этом, постараться сделать фотографию максимально четкой;
2) проверить результаты работы программы. Для этого пользователь смотрит на полученную фотографию. Если фотография получилась четкой, то контуры будут определены. В противном случае, или в случае погрешности при определении контуров, пользователь может вернуться к созданию фотографии.

Создание фотографии происходит следующим образом: при запуске приложения на экране сразу отображается вид с камеры телефона. Для создания фото необходимо нажать в любом месте рабочей области приложения (рисунок 3.3).

Рисунок 3.3 – Пример работы камеры приложения

Затем происходит переход к следующему окну приложения. В этом окне показаны фотография, с обозначенными на ней контурами, измерения, которые соответствуют измерениям линейки Поляка, и вероятность глаукомы у исследуемого пациента (рисунок 3.4).

Рисунок 3.4 – Интерфейс приложения

Если фотография получилась слишком нечеткой, то приложение сообщит об этом и попросит сделать новое фото (рисунок 3.5).

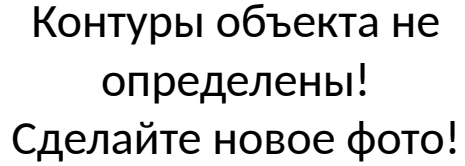


Рисунок 3.5 – Интерфейс приложения при ошибке

Если пользователю потребовалось сделать новую фотографию, то для этого ему необходимо нажать на кнопку «Назад», которая есть у каждого телефона (рисунок 3.6).

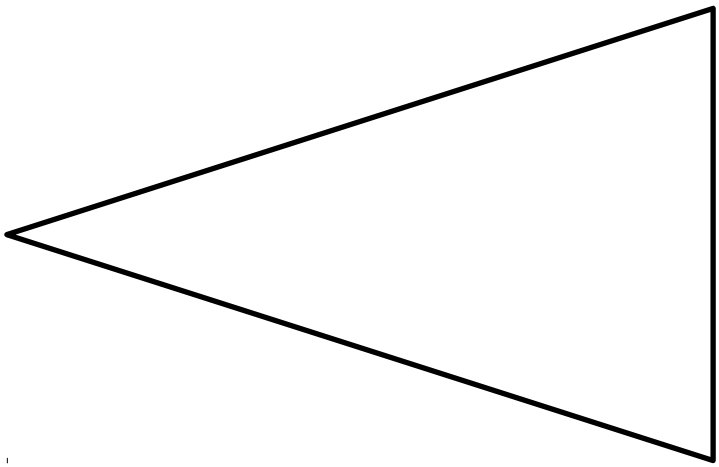


Рисунок 3.6 – Кнопка «Назад»

3.4 Проверка работы программы на экспериментальных данных

Для установки приложения на телефон необходимо скачать приложение. После установки в меню появится ярлык для запуска приложения (рисунок 3.7).

Рисунок 3.7 – Иконка приложения и рабочий стол телефона

Затем, запускаем программу, нажав на иконку приложения. После запуска мы сразу видим готовую к работе камеру (рисунок 3.8).

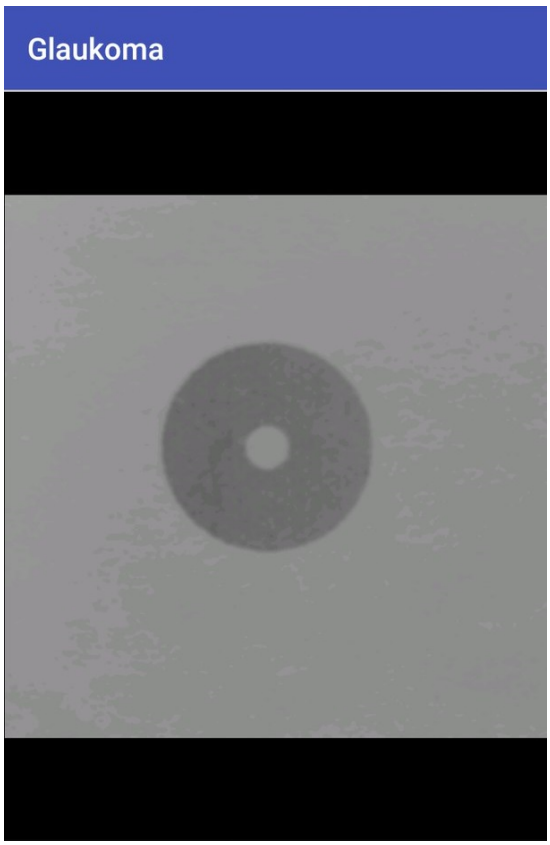


Рисунок 3.8 – Пример работы камеры приложения

Затем необходимо сделать фото. Для того, чтобы сделать фотографию, необходимо нажать в любом месте рабочей области приложения. После нажатия происходит переход на следующее окно приложения (рисунок 3.9).

В этом окне видно фотографию, которая была сделана. На ней отображаются круги, которые были найдены - их контуры обведены черным. Это сделано для того, чтобы пользователь мог понять, верно ли были найдены контуры. Также под фото показано значение внутриглазного давления по линейке Поляка и вероятность глаукомы у пациента (рисунок 3.10).

Если пользователя не устраивают найденные контуры, то он может вернуться к предыдущему окну, чтобы сделать новую фотографию (рисунок 3.11). К тому же, камера может не успеть сфокусироваться, что приведет к размытому снимку. Это не позволит точно определить контуры объекта, и приложение выдаст сообщение с просьбой сделать новую фотографию (рисунок 3.12).

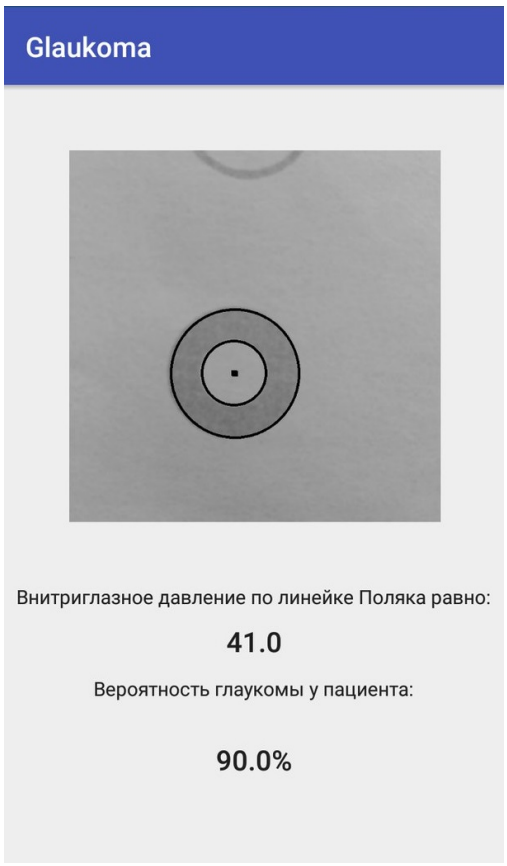


Рисунок 3.9 – Пример результатов работы программы

Рисунок 3.10 – Пример результатов работы программы

Рисунок 3.11 – Пример погрешности в результатах работы приложения

Рисунок 3.12 – Пример ошибки, допускаемой пользователем при работе с приложением

Если пользователю потребовалось сделать новую фотографию, то для этого ему необходимо нажать на кнопку «Назад», которая есть у каждого телефона (рисунок 3.13).

Рисунок 3.13 – Кнопка «Назад»

3.5 Выводы по разделу

В результате были разработаны алгоритмы определения контуров окружностей и определения контуров кольца по результатам определения контуров окружностей. Они являются основными алгоритмами в приложении.

Также был разработан интерфейс приложения. Он похож на интерфейс стандартной камеры телефона, поэтому уже хорошо знаком пользователю. Отличие заключается только в том, что, помимо фотографии, на экране будут показаны результаты работы с изображением.

Кроме того, была проверена работа программы на экспериментальных данных. Были использованы различные радиусы внутренних кругов. Программа успешно определяла контуры кругов. Помимо этого, были проверены случаи, когда фотография была смазана и выдает ли ошибку приложения.

ЗАКЛЮЧЕНИЕ

Разработка приложения для определения глаукомы очень важна, так как заболевание это очень распространенное. Оно занимает одно из первых мест среди причин неизлечимой слепоты и имеет важнейшее социальное значение. После 40 лет каждому человеку необходимо проходить профилактический осмотр у врача-офтальмолога не реже 1-2 раза в год. Это говорит о важности развития в этой области. Данная работа предоставит простой инструмент для определения вероятности глаукомы у пациента, не затрачивая времени на измерения линейкой Поляка, что ускорит работу врача. Это позволит врачу принимать больше пациентов в день и поможем определить точное значение внутриглазного давления, в случае если оно слишком высокое

В разделе 2 была описана модель, по которой вычисляются контуры окружностей и их центры. Преобразование Хафа показало себя с наилучшей стороны во многих задачах компьютерного зрения. К тому же, частный случай реализации данного алгоритма как раз решает задачу нахождения 3 параметров окружности: радиус и x, y координаты центра окружности.

В ходе работы был разработан интерфейс пользователя программы. Он знаком пользователю, так как по такому же принципу работают камеры телефона: пользователь делает фотографию, затем видит, что получилось. Если фотография «не подходит», то он возвращается обратно к камере и делает новое фото.

В результате работы программы пользователь получает несколько значений.

Первое – значение, которое соответствует измерению линейки Поляка. Это сделано для того, чтобы пользователь сам определял диагноз. К тому же, в разное время суток внутриглазное давление разное, и с помощью данного значения можно узнать динамику. Второе – вероятность глаукомы у пациента. Данное значение является больше подсказкой пользователю, чем окончательным диагнозом. Но, исходя из этой информации, уже можно сделать определенные выводы.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Горбань, А.И. Исследование поля зрения и внутриглазного давления у взрослых и детей/ А.И. Горбань – Л.: ЛПМИ, 1982. – 95 с.
2. Техника тонометрии по Маклакову; основы клинической трактовки результатов исследования – Дата обновления: 30.10.2011. URL: http://zreni.ru/1235-tehnika-tonometrii-po-maklakovu-osnovy-klinicheskoy-traktovki-rezultatov-issledovaniya.html (дата обращения: 25.01.2017)
3. Клинические лекции по офтальмологии – Дата обновления: 23.11.2012. URL: http://www.nnre.ru/medicina/klinicheskie_lekcii_po_oftalmologii/ (дата обращения: 25.01.2017).
4. Контурный анализ – детектирование зашумленного бинарного объекта – Дата обновления: 08.10.2015. URL: http://recog.ru/blog/opencv/229.html (дата обращения: 08.01.2017).
5. OpenCV шаг за шагом. – Дата обновления: 20.08.2011. URL: http://robocraft.ru/page/opencv// (дата обращения: 25.01.2017).
6. Bradski, G. Learning OpenCV /G. Bradski, A.Kaehler – Sebastopol: O'Reilly Media, 2008. – 580 p.
7. Baggio, D.L. Mastering OpenCV with Practical Computer Vision Projects/ D.L. Baggio, S.Emami, D.M.Escriva - Birmingham: Packt Publishing, 2012. – 318 p.
8. Muhammad,A. OpenCV Android Programming By Example/ Muhammad Amgad – Birmingham: Packt Publishing, 2016. – 171 p.
9. Joshi,P. OpenCV By Example/ Joshi Prateek, Escriva David Millan, Godoy Vinicius – Birmingham: Packt Publishing, 2016. – 296 p.
10. Android Development with OpenCV – Дата обновления: 30.10.2011. URL: http://docs.opencv.org/2.4/doc/tutorials/introduction/android_binary_package/dev_with_OCV_on_Android.html (дата обращения: 25.01.2017)
11. Добавление библиотеки OpenCV в проект Android Studio – Дата обновления: 08.07.2015. URL: https://habrahabr.ru/post/262089/ (дата обращения: 08.01.2017)
12. Распознавание образов с OpenCV: Контуры против Haatrainig – Дата обновления: 29.03.2013. URL: https://habrahabr.ru/post/174703/ (дата обращения: 08.01.2017)
13. Сегментация изображения в OpenCV с помощью алгоритмов Watershed и Distance Transform – Дата обновления: 11.08.2015. URL: http://techcave.ru/posts/54-segmentacija-izobrazhenija-v-opencv-s-pomoschyu-algoritmov-watershed-i-distance-transform.html (дата обращения: 08.01.2017)
14. Прэтт, У. Цифровая обработка изображений/ У. Прэтт – М.: Мир, 1982. – 480 с.
15. Гонсалес, Р. Цифровая обработка изображений. / Р. Гонсалес – М.: Техносфера, 2005. – 1072 с.
16. Местецкий, Л.М. Математические методы распознавания образов. / Л.М. Местецкий – М.: МГУ, 2004. – 144 с.

17. Какие языки программирования нужно знать– Дата обновления: 04.02.2016. URL: https://trashbox.ru/topics/94183/ kakie-yazyki-programmirovaniya-nuzhno-znat (дата обращения: 08.01.2017)
18. Howse, J. Android Application Programming with OpenCV / J. Howse – Birmingham: Packt Publishing, 2013. – 115 p.
19. Ретабоуил, С. Android NDK. Разработка приложений под Android на C/C++ / С. Ретабоуил – Москва: ДМК Пресс, 2012. – 496 с.
20. Программирование под ОС Андроид. – Дата обновления: 23.01.2017. URL: http://metanit.com/java/android/ (дата обращения: 25.01.2017).

ТЕКСТ ПРОГРАММЫ

Модуль MainActivity

```
// файл java
package c.myapplication;

import android.content.Intent;
import android.graphics.Bitmap;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.SurfaceView;
import android.view.View;

import org.opencv.android.BaseLoaderCallback;
import org.opencv.android.CameraBridgeViewBase;
import org.opencv.android.JavaCameraView;
import org.opencv.android.LoaderCallbackInterface;
import org.opencv.android.OpenCVLoader;
import org.opencv.android.Utils;
import org.opencv.core.Core;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.Point;
import org.opencv.core.Scalar;
import org.opencv.core.Size;
import org.opencv.imgproc.Imgproc;

import java.io.ByteArrayOutputStream;


public class MainActivity extends AppCompatActivity implements
CameraBridgeViewBase.CvCameraViewListener2 {

    private static String TAG = "MainActivity";

    //работа с камерой
    JavaCameraView javaCameraView;
    Mat mRgba;
    Mat mRgbaF; //для поворота камеры
```

```
Mat mRgbaT; //для поворота камеры

//детектор границ
Mat imgGray, imgCanny;

//здесь мы храним найденные круги
Mat circles;

//изображение imgCanny
Bitmap bmp=null;

//радиус кружка сплющивания
double rFlat;

BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback(this) {
   @Override
   public void onManagerConnected(int status) {
      switch(status){
         case BaseLoaderCallback.SUCCESS:{
            javaCameraView.enableView();
            break;
         }
         default:{
            super.onManagerConnected(status);
            break;
         }
      }
   }
};

@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   //Определение камеры
                                    javaCameraView = (JavaCameraView)
findViewById(R.id.java_camera_view);
   javaCameraView.setVisibility(SurfaceView.VISIBLE);
   javaCameraView.setCvCameraViewListener(this);

   //обработчик нажатия
   javaCameraView.setOnClickListener (new View.OnClickListener() {
      @Override
      public void onClick(View v) {
```

```
                findMyCircle();
                //Создание картинки
                        bmp = Bitmap.createBitmap(imgCanny.cols(), imgCanny.rows(),
Bitmap.Config.ARGB_8888);
                Utils.matToBitmap(imgCanny, bmp);
                Intent intent = new Intent(MainActivity.this, CheckPhoto.class);

                //преобразование картинки к массиву байтов
                ByteArrayOutputStream stream = new ByteArrayOutputStream();
                bmp.compress(Bitmap.CompressFormat.PNG, 0, stream);
                byte[] byteArray = stream.toByteArray();

                //переход на следующий экран
                intent.putExtra("picture", byteArray);
                intent.putExtra("radius", rFlat);
                startActivity(intent);
            }});
        }

        public void findMyCircle(){
            // Значение аккумулятора
            double dp = 1.2d;
            // Минимальная дистанция между центрами разных окружностей
            double minDist = 1;

            // максимальный и минимальный радиусы окружностей
            int minRadius = 1, maxRadius = 400;

            // param1 = значение градиента
            // param2 = критерий фильтрации
              //Чем меньше значение param2, тем больше будет найдено кругов (в том
числе и "ложных").
            //Чем больше значение param2, тем больше кругов будет отсеяно.
            int numberOfCircles=100;
            double param1 = 30, param2 = 30;

            //Убираем "ненужные" круги
            while (numberOfCircles>10){
               if(numberOfCircles<300){
                  param1+=2;
                  param2+=2;
               }else{
                  param1+=10;
                  param2+=10;
```

```
            }
            /* Находим окружности на изображении */
                                        Imgproc.HoughCircles(imgCanny,     circles,
Imgproc.CV_HOUGH_GRADIENT,
                  dp, minDist, param1, param2, minRadius, maxRadius);

            /* Считаем количество найденных кругов */
            numberOfCircles = (circles.rows() == 0) ? 0 : circles.cols();
         }
         double min=400, max=1;
         int[] iminmax={0,0};

         //Находим кольцо
         for (int i=0; i<numberOfCircles; i++) {

            double[] circleCoordinates = circles.get(0, i);
            int radius = (int) circleCoordinates[2];
            if(radius<min){
               min=radius;
               iminmax[0]=i;
            }
            if(radius>max){
               max=radius;
               iminmax[1]=i;
            }
         }
         rFlat=min/max;

         if(numberOfCircles>1) numberOfCircles=2;

         for(int i=0; i<numberOfCircles; i++) {
             /* получаем детальную информацию о кругах, circleCoordinates[0, 1, 2]
= (x,y,r)
            * (x,y) - координаты центров кругов
            */
            double[] circleCoordinates = circles.get(0, iminmax[i]);
            int x = (int) circleCoordinates[0], y = (int) circleCoordinates[1];
            Point center = new Point(x, y);
            int radius = (int) circleCoordinates[2];



            /* Рисуем оркужность */
            Imgproc.circle(imgCanny, center, radius, new Scalar(0, 255, 0), 4);
```

```
            /* Рисуем центр */
            Imgproc.rectangle(imgCanny, new Point(x - 5, y - 5),
                  new Point(x + 5, y + 5),
                  new Scalar(0, 128, 255), -1);
         }
      }

      @Override
      protected void onPause() {
         super.onPause();
         if (javaCameraView != null)
            javaCameraView.disableView();
      }

      @Override
      protected void onDestroy() {
         super.onDestroy();
         if (javaCameraView != null)
            javaCameraView.disableView();
      }

      @Override
      protected void onResume(){
         super.onResume();
         if(OpenCVLoader.initDebug()){
            Log.i(TAG, "OpenCV loaded");
             mLoaderCallback.onManagerConnected(LoaderCallbackInterface.SUCCES
S);
         }
         else{
            Log.i(TAG, "OpenCV not loaded");
                OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_3_2_0,
this, mLoaderCallback); //Вызываем повторно OpenCV
         }
      }

      @Override
      public void onCameraViewStarted(int width, int height) {
         //отображение камеры и решение проблемы "пьяной камеры"
         mRgba = new Mat(height,width, CvType.CV_8UC4);
         mRgbaF = new Mat(height, width, CvType.CV_8UC4);
         mRgbaT = new Mat(width, width, CvType.CV_8UC4);
```

```
        //детектор границ
        imgGray = new Mat(height,width, CvType.CV_8UC1);
        imgCanny = new Mat(height,width, CvType.CV_8UC1);

        /* создаем Mat-объект для хранения информации об окружностях */
        circles = new Mat(height,width, CvType.CV_8UC1);
    }

    @Override
    public void onCameraViewStopped() {
        mRgba.release();
    }

    @Override
        public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame
inputFrame) {
        mRgba = inputFrame.rgba();
        // Поворачивает mRgba на 90 градусов
        Core.transpose(mRgba, mRgbaT);
        Imgproc.resize(mRgbaT, mRgbaF, mRgbaF.size(), 0,0, 0);
        Core.flip(mRgbaF, mRgba, 1 );

        //Использование детекторов
        Imgproc.cvtColor(mRgba,imgGray,Imgproc.COLOR_RGB2GRAY); //Серый
        //Imgproc.Canny(imgGray, imgCanny, 50, 150);
        Imgproc.GaussianBlur(imgGray, imgCanny, new Size(9, 9), 2, 2);

        return imgCanny;
    }
}
```

Модуль CheckPhoto

```
// файл java
package c.myapplication;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageView;
import android.widget.TextView;
```

```
public class CheckPhoto extends AppCompatActivity {

    //радиус кружка сплющивания
    double radius;
    //вес гири
    double P=10000;

    TextView textViewNumber;
    TextView textViewTextNumber;
    TextView textViewProbability;
    TextView textViewTextProbability;
    double[][] RulerPolyak={
            {12, 1,56},{12,0.982,56},{12,0.971,56},{12,0.958,56},
            {13,0.945,50},{13,0.933,50},{13,0.922,50},{13,0.911,50},
            {14,0.899,42},{14,0.887,42},{15,0.876,36},{15,0.865,36},
            {16,0.853,28},{16,0.841,28},{16,0.829,28},{17,0.816,20},
            {17,0.806,20},{18,0.794,10},{18,0.781,10},{19,0.768,1},
            {20,0.757,1},{20,0.745,1},{21,0.721,1},{22,0.710,5},
            {22,0.699,5},{23,0.688,5},{24,0.676,5},{25,0.665,10},
            {25,0.654,10},{26,0.642,20},{27,0.631,28},{28,0.618,36},
            {29,0.606,42},{30,0.593,50},{31,0.582,56},{32,0.570,64},
            {34,0.558,72},{35,0.547,80},{36,0.535,83},{38,0.525,86},
            {39,0.513,88},{41,0.502,90},{43,0.489,91},{45,0.479,92},
            {47,0.465,93},{49,0.454,94},{51,0.441,95},{54,0.431,96},
            {56,0.418,97},{59,0.406,98},{62,0.394,99},{65,0.379,99},
            {69,0.368,99},{73,0.356,99},{77,0.344,99},{82,0.333,99},
            {87,0.322,99},{92,0.310,99},{98,0.298,99},{105,0.289,99}
            };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_check_photo);

       ImageView imageView = (ImageView) findViewById(R.id.imageView);

       //помещаем картинку на экран
       byte[] imgBytes = getIntent().getByteArrayExtra("picture");
                  Bitmap bmp = BitmapFactory.decodeByteArray(imgBytes, 0,
imgBytes.length);
       imageView.setImageBitmap(bmp);
       radius=getIntent().getDoubleExtra("radius", -1);

       textViewNumber = (TextView) findViewById(R.id.textViewNumber);
```

```
        textViewProbability = (TextView) findViewById(R.id.textViewProbability);
                                                textViewTextNumber = (TextView)
findViewById(R.id.textViewTextNumber);
                            textViewTextProbability = (TextView) findViewById(R.id.
textViewTextProbability);

        if(radius==-1){
           textViewNumber.setText("Ошибка приложения:");
           textViewTextNumber.setVisibility(View.INVISIBLE);
           textViewProbability.setText("Свяжитесь с разработчиком");
           textViewTextProbability.setVisibility(View.INVISIBLE);
        }
        else{
           if(radius==400){
              textViewNumber.setText("Объект не определен:");
              textViewTextNumber.setVisibility(View.INVISIBLE);
              textViewProbability.setText("сделайте новое фото!");
              textViewTextProbability.setVisibility(View.INVISIBLE);
           }
           else{
              for(int i=RulerPolyak.length-1;i>-1;i--){
                 if(RulerPolyak[i][1]>=radius){
                    if(i==RulerPolyak.length-1){
                       textViewNumber.setText(">105");
                       textViewProbability.setText("99%");
                    }
                    else{
                       textViewNumber.setText(String.valueOf(RulerPolyak[i][0]));
                            textViewProbability.setText(String.valueOf(RulerPolyak[i][2])
+"%");
                    }
                    break;
                 }
                 else{
                    if(i==0){
                       textViewNumber.setText("<12");
                       textViewProbability.setText(">56%");
                    }
                 }
              }

           }
        }
```

```
    }
}
```

Модуль activity_main

```
// файл xml
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="c.myapplication.MainActivity">

    <org.opencv.android.JavaCameraView
        android:id="@+id/java_camera_view"
        android:layout_width="361dp"
        android:layout_height="497dp"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="49dp"
        app:layout_constraintBottom_toBottomOf="parent"
        tools:layout_editor_absoluteX="4dp"
        tools:layout_editor_absoluteY="6dp" />


</android.support.constraint.ConstraintLayout>
```

Модуль activity_check_photo

```
// файл xml
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="c.myapplication.CheckPhoto">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="267dp"
        android:layout_height="329dp"
```

```
    app:srcCompat="@android:drawable/alert_light_frame"
    android:layout_marginLeft="8dp"
    app:layout_constraintLeft_toLeftOf="parent"
    android:layout_marginRight="8dp"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    android:layout_marginTop="16dp"
    app:layout_constraintHorizontal_bias="0.516" />

<TextView
    android:id="@+id/textViewTextNumber"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="13dp"
    android:text="Внитриглазное давление по линейке Поляка равно:"
    android:textAppearance="@style/TextAppearance.AppCompat.Body1"
    android:visibility="visible"
    app:layout_constraintTop_toBottomOf="@+id/imageView"
    android:layout_marginRight="8dp"
    app:layout_constraintRight_toRightOf="parent"
    android:layout_marginLeft="8dp"
    app:layout_constraintLeft_toLeftOf="parent" />

<TextView
    android:id="@+id/textViewTextProbability"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="79dp"
    android:text="Вероятность глаукомы у пациента:"
    android:textAppearance="@style/TextAppearance.AppCompat.Body1"
    android:visibility="visible"
    app:layout_constraintTop_toBottomOf="@+id/imageView"
    android:layout_marginRight="8dp"
    app:layout_constraintRight_toRightOf="parent"
    android:layout_marginLeft="8dp"
    app:layout_constraintLeft_toLeftOf="parent" />

<TextView
    android:id="@+id/textViewNumber"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="0dp"
    android:text="000"
    android:textAppearance="@style/TextAppearance.AppCompat.Body2"
```

```
        android:textSize="20sp"
        android:visibility="visible"
        app:layout_constraintTop_toBottomOf="@+id/textViewTextNumber"
        android:layout_marginBottom="8dp"
        android:layout_marginRight="8dp"
        app:layout_constraintRight_toRightOf="parent"
        android:layout_marginLeft="8dp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintHorizontal_bias="0.501"
        app:layout_constraintVertical_bias="0.764"
        app:layout_constraintBottom_toTopOf="@+id/textViewTextProbability" />

    <TextView
        android:id="@+id/textViewProbability"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginTop="0dp"
        android:text="000"
        android:textAppearance="@style/TextAppearance.AppCompat.Body2"
        android:textSize="20sp"
        android:visibility="visible"
        app:layout_constraintHorizontal_bias="0.501"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textViewTextProbability"
        app:layout_constraintVertical_bias="0.345"
        app:layout_constraintBottom_toBottomOf="parent" />
</android.support.constraint.ConstraintLayout>
```

Модуль AndroidManifest

```
// файл xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="c.myapplication">

    <uses-permission android:name="android.permission.CAMERA" />
    //Права на использование камеры
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
```

```
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".CheckPhoto"></activity>
    </application>

</manifest>
```

Модуль build

```
// файл gradle
apply plugin: 'com.android.application'

android {
    compileSdkVersion 25
    buildToolsVersion "25.0.2"
    defaultConfig {
        applicationId "c.myapplication"
        minSdkVersion 14
        targetSdkVersion 25
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
                proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-
rules.pro'
        }
    }
}

dependencies {
```

```
    compile fileTree(include: ['*.jar'], dir: 'libs')
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile project(':OpenCV')
    compile 'com.android.support:appcompat-v7:25.3.0'
    compile 'com.android.support.constraint:constraint-layout:1.0.2'
    testCompile 'junit:junit:4.12'
}
```