Оглавление

Введение

Обнаружение лица и отслеживание движения (трекинг) глаз позволяет получать ценную информацию, которую можно применять для решения широкого диапазона задач. Относительное положение глаз можно отслеживать с помощью специальных устройств, но их высокая стоимость и функциональные ограничения делают эти существующие решения непривлекательными или даже непригодными для использования на изображениях глаз, полученных со стандартных компьютерных камер с невысоким разрешением.

Целью данной диссертации является

Выбор минимальных аппаратных и разработка программных средств, обеспечивающих отслеживание взгляда в процессах человеко-машинного взаимодействия программной инженерии. Для достижения этой цели решаются задачи анализа подходов к отслеживанию взгляда, выявления требований к видеоаппаратуре, разработки алгоритмов анализа видеопотока и реализации этих алгоритмов в контексте пользовательского интерфейса (UI) компьютерных приложений.

Условием решения указанных задач оказалось исследование проблем(ы) оперативного определения цветовой гаммы элементов глазного яблока в видеопотоке от ординарной web-камеры.

Исследования проведены и поставленные задачи успешно решены. Результаты отражены в пояснительной записке, содержащей 68 стр., 17 рисунков.

В главе 1 рассмотрены технологии распознавания и направления взгляда. А так же будет проведён анализ проблем анализ проблем распознавания и направления взгляда.

В главе 2 проведён анализ перспективных областей применения технологий распознавания направления взгляда. И на основе этого анализа обоснованы требований к этим технологиям.

В главе 3 представлена архитектура библиотеки типов и основные алгоритмические решения по предлагаемой технологии распознавания направления взгляда.

В главе 4 определены состав и организация тестовых примеров, демонстрирующих предлагаемую библиотеку типов.

В главе 5 приведены итоги тестирования и сделаны выводы

В заключении подведены итоги проделанной работы.

Приложении содержат текст разработанных программных средств.

1. Обзор технологий и анализ проблем распознавания направления взгляда

1.1 Определения и условные обозначения (сокращения)

взаимодействие между человеком и машиной (англ. HMI – Human Machine Interaction)

Пользовательский интерфейс – интерфейс между технической системой и её пользователем (англ. UI – User Interface)

Графический (пользовательский) интерфейс – пользовательский интерфейс, реализуемый на основе графического дисплея, клавиатуры и манипуляторов (англ. GUI – Graphic User Interface)

Распознавание глаз – операция устройства или программы, выдающая в результате применения к изображению координаты областей, которые могут человеком интерпретироваться как пара глаз на лице (англ. eye recognition)

Отслеживание глаз – операция устройства или программы, выдающая в результате применения к последовательности изображений траекторию областей, которые могут человеком интерпретироваться как пара глаз на лице (англ. eye tracking)

Взгляд – состояние зрительной системы, обеспечивающее восприятие информации в определённой точке пространства или по определённому направлению (англ. gaze)

Распознавание взгляда – операция устройства или программы, выдающая в результате применения к изображению координаты точки пространства, на которую направлен взгляд, или направление взгляда (англ. gaze recognition)

Отслеживание взгляда – операция устройства или программы, выдающая в результате применения к последовательности изображений («видеопотоку») траекторию точки пространства, на которую направлялся взгляд, или последовательность направлений взгляда (англ. gaze tracking)

Айтрекер (англ. Eye-tracker) — устройство, используемое для определения ориентации оптической оси глазного яблока в пространстве (то есть для распознавания глаз).

светодиодных источниках освещения.

1.4. Перспективные области применения технологий распознавания направления взгляда

1.4.1 Айтрекинг в последние годы

В последние годы технология отслеживания глаз быстро развивалась с улучшением стабильности, точности и частоты дискретизации. Приобретение стартапов по отслеживанию глаз крупными компаниями, а также развертывание программного обеспечения и нескольких устройств, поддерживающих отслеживание глаз, также растет. Недавно компании, выпускающие гарнитуры виртуальной реальности, вложили значительные средства в технологию отслеживания взгляда, чтобы продвинуть ее дальше и создать захватывающий опыт. Facebook и Google приобрели стартапы по отслеживанию глаз, Eye Tribe и Eyefluence соответственно, и, как ожидается, будут внедрять эту технологию в свои будущие приложения. Проект Fove, финансируемый с KickStarter, - это первая гарнитура VR (виртуальной реальности) со встроенным отслеживанием взгляда. Инвестирование и финансирование новых стартапов и приобретение компаний повышает спрос на технологию отслеживания движения глаз.

Оскар Вернер, вице-президент Tobii Tech, упоминает два основных преимущества датчиков, используемых в технологии отслеживания взгляда. Он говорит: «Во-первых, он информирует устройство о том, что интересует пользователя в любой момент времени. Во-вторых, он предоставляет дополнительный способ взаимодействия с контентом, не убирая ничего лишнего. Это означает, что увеличивается пропускная способность связи между пользователем и устройством».

Одно из наиболее широко используемых в настоящее время приложений для технологии отслеживания взгляда - определение того, как пользователи обычно взаимодействуют с Интернетом и веб-страницами. Лучшее понимание этого может помочь фирмам и брендам в максимальном воздействии на пользователя и сделать их опыт максимально удобным.[12]

1.4.2 Потребительские устройства

В будущем отслеживание взгляда станет стандартной функцией ноутбуков, настольных мониторов и смартфонов нового поколения, что создаст основу для огромной переоценки того, как устройства общаются с нами или мы с ними общаемся. Мобильные телефоны, такие как iPhone X или Xiaomi Mi8, и ноутбуки с обращенными к пользователю камерами с датчиком глубины можно модернизировать для запуска трехмерного отслеживания взгляда без необходимости в дополнительном устройстве отслеживания взгляда. Затем камера с датчиком глубины активируется для обнаружения внимания пользователя к частям экрана, а также позволяет пользователю частично перемещаться по устройству с помощью глаз. Для игровых ноутбуков можно использовать 3D-взгляд или отслеживание взгляда, чтобы улучшить игровой процесс при взаимодействии с аватарами или затемнить информационные меню, которые находятся не в фокусе, чтобы убрать беспорядок на экране. В дополнение к этому, в ближайшем будущем

появятся и другие приложения отслеживания взгляда с помощью 3D-технологии. [12]

1.4.3 Автомобильная промышленность

Мониторинг водителя

Как правило, мониторинг водителя считается главной мерой для обеспечения безопасности при вождении транспортных средств. Однако Европейская программа оценки новых автомобилей (NCAP) предполагает, что при использовании технологии отслеживания взгляда оценка будет развиваться в зависимости от того, насколько точно и надежно отслеживается статус вождения водителя. Программа войдет в применение к 2020 году. Таким образом, с помощью технологии трехмерного отслеживания движения глаз, обнаружение внимания водителя к контролю и обеспечение скоординированных поставок в полуавтономных автомобилях будет легким, предлагая пользователям безопасное решение.

Виртуальный второй пилот

В последние годы автомобили и другие транспортные средства также взаимодействуют с водителем на основе голосовых и жестовых команд. Тем не менее, восприятие внимания отставало. В этом отношении помогает айтрекинг, которое может не только позволить автомобильному ассистенту ощущать взгляд водителя и пассажиров на такие части салона автомобиля, как центральный экран автомобиля (в некотором количестве современных автомобилей помимо радио в центре автомобиля стоит интерактивный экран), приборная панель со спидометром и зеркала заднего вида, но и обеспечивать естественное взаимодействие с AR (дополненной реальностью), индикаторами на лобовом стекле и даже позволяют виртуальному второму пилоту обнаруживать взгляд на объекты за пределами транспортного средства или автомобиля.[12]

1.4.4 Сектор розничной торговли

В секторе розничной торговли, наблюдая за вниманием покупателя, продавцы могут анализировать на сколько продукт интересует покупателей. Регистрируя данные о движении глаз, когда покупатели перемещаются по магазинам или просматривают продукт на полках, маркетологи могут получить подлинную информацию и динамическое понимание того, что действительно привлекает потребителей. Без необходимости калибровки 3D-отслеживание взгляда может предложить аналитику внимания, которая определяет количество просмотров и точек фокусировки. Компании, занимающиеся маркетинговыми исследованиями, могут затем использовать эту информацию, чтобы предложить розничным продавцам оптимизацию в отношении их подхода к продажам и качества обслуживания клиентов. [12]

1.4.5 Здравоохранение

В последние годы в сфере здравоохранения использование айтрекинга стало тенденцией. Многие исследователи начали использовать технологию отслеживания движения глаз для исследований безопасности пациентов. Технология также активно используется в специализированных медицинских центрах для создания системы общения с пациентами, полностью или частично утратившими двигательные функции, страдающими серьезными поражениями опорно-двигательного аппарата или с нарушениями речи. Для таких пациентов важна как психологическая, так и социальная реабилитация. По этой причине общение для них играет важную роль.

Использование отслеживания взгляда в смоделированных условиях может помочь нам лучше понять, какие источники информации пользователи используют, а какие не используют при выполнении рутинных процессов. В частности, данные, полученные в результате отслеживания взгляда, могут

использоваться для оценки общих процессов, подверженных ошибкам. Информация о поведении глаз экспертов может привести к разработке протоколов обучения для помощи в обучении студентов и начинающих практиков. Технология айтрекинга может изменить способ использования множества устройств в ближайшем будущем.[12]

1.4.6 Глаза вместо «мышки»

Еще один удачный пример практического применения отслеживания взгляда с помощью ИИ-алгоритмов — решение, которое позволяет предугадать, что человек хочет увидеть. Это совместный проект компаний SAP и 4tiitoo. Их разработка представляет собой ПО, позволяющее отслеживать движение глаз по монитору компьютера таким образом, чтобы определять главный предмет интереса смотрящего без ввода ключевых слова поискового запроса.

Когда человек ищет что-либо, глаза сканируют пространство, а мозг подсознательно фильтрует интересующий его контент. Разработанная в рамках проекта платформа NUIA с помощью ИИ может вычислять интересующие человека предметы без слов. Трекер фиксирует движение глаз по монитору и определяет уровни интереса человека к различным предметным областям. Если взгляд задерживается на изображении города дольше, чем на картинке с автомобилем, программа предложит продолжить поиск в выбранном вами направлении, автоматически предлагая списки потенциально интересных тем и предоставляя выдачи поисковика согласно всем изменениям маршрута движения ваших глаз.

В качестве теста NUIA помогает найти в базе из 5 тыс. снимков разных людей фотографию конкретного человека. Участник эксперимента запоминает фото человека, которого он хочет найти. Далее ПО предлагает ему для старта весь каталог изображений, при этом фиксируя, куда

направлен взгляд пользователя, и просчитывает «уровень интереса» к различным характеристикам внешности: цвет волос, пол, возраст и т. д. Пошагово происходит фильтрация и отсев тысяч снимков — вплоть до искомого. Процесс занимает около 2 минут.

Технология позволяет персонифицировать механизмы поиска в Интернете. Это способно дать толчок дополнительным продажам в электронной коммерции, развитию офисных программ, а также может полностью изменить принципы взаимодействия компьютера с человеком. Например, вполне вероятен отказ от привычных манипуляторов типа компьютерной мыши, стилуса или дизайнерского планшета. [13]

Так же областями применения являются:

- когнитивные исследования
- медицинские исследования
- человеческий фактор
- юзабилити исследования
- исследования процесса перевода с одного языка на другой
- симуляция транспортных средств
- исследования в реальных транспортных средствах
- технологии виртуальной реальности
- исследование взрослых пациентов
- исследования подростков
- исследование пожилых пациентов
- исследования обезьян
- спортивные тренировки
- регистрация в fMRI/MEG/EEG

- коммерческий айтрекинг (веб юзабилити, реклама, маркетинг, банкоматы)
- оптимальный выбор одного из нескольких вариантов
- коммуникационные системы для полностью парализованных людей
- улучшенные системы видеокоммуникации
- детекция жизнеспособности[14][15][16]

1.5 Проблемы распознавания взгляда и актуальность развития технологий распознавания взгляда применительно к программной инженерии

Из-за временной нестабильности потенциалов сигналов ЭОГ и длительности саккад, становится затруднительным использование ЭОГ для измерения медленных движений глаз и определения позиции взгляда. Однако ЭОГ является весьма устойчивой техникой определения саккадического движения глаз, связанного с изменением направления взгляда, а также детекции моргания глаз. В противоположность методам, основанным на видеозаписи, ЭОГ позволяет проводить регистрацию движений глаз даже когда глаза закрыты и, таким образом, ЭОГ может быть использована в исследованиях процесса сна. Это весьма малоресурсоемкий подход, который в противоположность методам, основанным на видеозаписи, не требует мощного компьютера, работает при различных световых условиях и легко может быть выполнен в виде мобильного устройства[17]. Таким образом, этот метод хорош для мобильного айтрекинга в повседневных ситуациях, а также в исследованиях стадии быстрых движений глаз во время сна.

Айтрекеры определяют ориентацию глазного яблока относительно некоторой системы координат. Если айтрекер монтируется на голову испытуемого, например, как в системе, основанной на ЭОГ, то необходимо заложить компенсацию движения головы испытуемого относительно этой системы координат. Вследствие этого, задача по определению точки взгляда испытуемого усложняется. В случае, если айтрекер неподвижно закреплен, то расчёт точки взора приводит к меньшим вычислительным затратам. Во многих системах голова испытуемого фиксируется с помощью офтальмологической рамки, вследствие этого становится возможным избежать дополнительных вычислений, связанных с движением головы испытуемого. В других системах выполняется компенсация движения головы с помощью магнитных сенсоров или дополнительного анализа видеоизображения.

Для устройств, монтируемых непосредственно на голове испытуемого, позиция головы и её ориентация в пространстве складываются с вектором направления взгляда человека. Для систем с неподвижным айтрекером направление головы вычитается из направления взгляда для того, чтобы определить позицию глаз на лице.

Информация о механизме и динамике движения глазного яблока пользуется большим спросом в научных исследованиях, однако, в большинстве случаев конечной задачей отслеживания глаз является определение точки взгляда, то есть отслеживание взгляда.[18]

Одной из трудностей при оценке систем айтрекинга является то, что глаз испытуемого крайне редко находится в неподвижном состоянии, бывает крайне трудно оценить небольшие, но крайне быстрые и иногда хаотические движения, связанные с воздействием источника шума внутри самого механизма систем айтрекинга (например цифровые шумы цифровых камер). Одним из полезных приёмов борьбы с этим эффектом является параллельная

запись двух глаз испытуемого и проверка позиции одного глаза по другому глазу. Глаза здорового человека очень хорошо связаны между собой, и разница в направлении оптических осей в вертикальном направлении обычно не превышает ± 2 угловых минут. Правильно функционирующая и чувствительная система айтрекинга должна показать эту степень согласованности глаз испытуемого. Любое возникновение более высокой угловой разницы может рассматриваться как ошибка измерения.[18]

Конечный потребитель может быть заинтересован, например, какие именно фрагменты изображения привлекли внимание испытуемого. Важным моментом является то, что айтрекер не может провести точное определение точки, которая привлекла внимание испытуемого. Тем не менее, айтрекинг весьма эффективен для определения примерной последовательности точек, привлекающих интерес. Для того, чтобы определить точку взгляда испытуемого, необходимо провести процедуру калибровки. В ходе подобных процедур, испытуемому предлагается последовательно направлять свой взгляд на серию калибровочных маркеров. Параллельно с этим айтрекер записывает координаты зрачка, которые соответствуют каждой из позиций калибровочных маркеров. Даже те техники, которые исследуют расположение сосудов на сетчатке глаза, не позволяют создать устройство, которое калибруется один-единственный раз на всех возможных испытуемых, поскольку, расположение сосудов на сетчатке является уникальным для каждого испытуемого. Точная и надежная калибровка необходима для получения правильных и воспроизводимых экспериментальных данных. Это может стать значительным препятствием при проведении экспериментов по отслеживанию глаз с испытуемыми с нестабильным взглядом.

Каждый метод айтрекинга имеет свои преимущества и недостатки и выбор оборудования для айтрекинга зависит от его стоимости и сферы применения. Существует офлайн и онлайн методы. Существует зависимость между ценой

и точностью системы. Большинство высокочувствительных систем стоят десятки тысяч долларов и требуют высокой квалификации персонала для настройки оборудования к экспериментам у конечного потребителя. Высокие темпы развития компьютерных технологий и технология обработки видео привели к возникновению относительно недорогих систем, которые пригодны в большинстве областей применения айтрекинга и простые в управлении. Интерпретация результатов все ещё требует некоторого уровня подготовки, кроме того, плохо откалиброванная система может привести к значительным погрешностям в процессе эксперимента.[18]

1.6 Выводы проведенного обзора

После проведенного обзора можно сделать следующие выводы:
1) На данный момент большинство решений по отслеживанию взгляда и распознанию глаз имеют множество недостатков:

- Дороговизна оборудования.
- Необходимость в стороннем оборудовании, носимом на голове, либо устанавливаемых в стороне
- Высокие требования к вычислительным мощностям ЭВМ, которая производит вычисления для отслеживания

2) Данная тема очень востребована, поскольку она может быть использована в различных жизненных сферах, таких как: розничная торговля, автомобильная промышленность, здравоохранение и человеко-машинное взаимодействие

3) Тема имеет большую актуальность, поскольку ведётся большое количество исследований, большие компании, такие как Google и Facebook инвестируют в эти исследования деньги и обычные потребители жертвуют/инвестируют свои деньги в проекты по отслеживанию взгляда

через такие сервисы как Kikcstarter.

2. Анализ перспективных областей применения технологий распознавания направления взгляда в программной инженерии и обоснования требований к этим технологиям.

2.1 Общие требования программной инженерии к технологиям отслеживания взгляда

Разрабатываемое программное решение должно быть простым в использовании и не требовательным к системным характеристикам ЭВМ. Для программного решения должно быть достаточно обыкновенной веб-камеры. Различные дополнительные средства для помощи в отслеживании, такие как электроды на голову, использование инфракрасных лучей, дополнительные аксессуары на голову крайне нежелательны. Поскольку в отслеживании из устройств присутствует только веб-камера, то точность измерений зависит только от её характеристик. Частота распознавания ограничена частотой кадров в секунду веб камеры.

Так же для нормального распознавания взгляда требуется, чтобы лицо пользователя было направлено прямо в веб-камеру. Направления лица в любую другую сторону может привести к некорректной работе программы.

2.2 Общие требования Требованияя к программному средству – «распознавателю взгляда» для программной инженерии

Рекомендуется разрабатывать данный программный комплекс по ГОСТ Р 51904-2002.

2.3. Требования к функциональным характеристикам

2.3.1. Общие функциональные требования

- параллельная работа пользователя с графическим интерфейсом и работа системы;

- входные параметры: изображения, на которых находится лицо пользователя, направленное прямо на камеру;

- выходные параметры: отфильтрованные изображения с местоположением белков и радужек, изображение с нарисованными найденными контурами белков и радужек, информация о направлении взгляда.

2.3.2. Требования к функциональным характеристикам каждой функции

- Функция конфигурирования фильтрацией. Предоставляет пользователю графический интерфейс для изменения параметров фильтрации первичного изображения.

- Функция ввода изображения. Функция просит пользователя ввести полный путь к изображению. В случае, если изображение не найдено или не может быть прочитано, функция просит ввести другой путь к изображению.

- Функция фильтрации изображения по заданным пользователем порогам. Функция берёт оригинальное изображение и в зависимости от порогов, заданных пользователем, фильтрует изображение соответственно

- Функция нахождения и отрисовки контуров. Функция ищет контуры на отфильтрованном пользователем изображении, а затем переносит эти контуры на оригинальное изображение

- Функция вывода всех изображений. Функция выводит два пороговых изображения: первое с порогами белков, второе с порогами радужек. Также

выводится оригинальное изображение, на котором отрисованы контуры пороговых изображений

- Функция поиска нужных HSV значений. Функция выводит оригинальное и пороговое изображение и окно со значениями для первичной фильтрации изображения. После завершения работы функции, параметры записываются в переменную и передаются на выход функции

- Функция перевода изображения из цветовой модели RGB в цветовую модель HSV. Функция берёт оригинальное изображение, переводит его в цветовую модель HSV и записывает его в переменную

- Функция поиска экстремумов на контуре. Функция производит поиск экстремальных точек на контуре и передаёт эти точки на выход

- Функция вывода экстремумов на изображение. Функция берёт найденные точки и отрисовывает их на оригинальном изображении

- Функция нахождения направления взгляда. Функция берёт всю полученную ранее информацию, анализирует её и в результате анализа делает выводы о том, в какую сторону направлен взгляд

2.4. Требования к надежности

2.4.1. Общие требования к надежности

- устойчивое функционирование;

- контроль входной/выходной информации: проверка на корректность входных параметров, обработка критических ситуаций и вывод информационных сообщений при возврате разных выходных данных;

- восстановление после отказа происходит в ручном режиме методом перезагрузки неисправного компонента системы;

• при проектировании ПО для увеличения отказоустойчивости предусматривается модульный подход;

• избыточность компонентов не предусматривается.

2.4.2. Классификация отказных ситуаций

Возможные отказные ситуации программы:

- белки глаз не были найдены на изображении;
- глаза не присутствуют на изображении вовсе;
- белки и кожа могут быть приблизительно одинаково из-за освещения или состояния здоровья пользователя;
- форма глаз, не позволяющая считать достаточное количество информации о белках и, в следствии, невозможность корректно определить направление взгляда;
- веб-камера стоит в плохих условиях, из за чего на изображении появляется большое количество цифровых шумов, которые мешают корректно определить направление взгляда.

2.5. Условия эксплуатации

• Носитель информации: любой энергонезависимый для хранения программы и любой энергозависимый для выполнения программы.

• Вид обслуживания: обслуживание не требуется.

Основное назначение программы при разработке – это бесконтактное управление ЭВМ. Определение направления взгляда может быть использовано для, например, управления экранной клавиатурой. Задержка взгляда в определенном направлении на несколько секунд позволяло бы управлять курсором на клавиатуре, а направление взгляда прямо засчитывается как сигнал о том, что выбранную букву надо ввести.

2.6. Требования к составу и параметрам технических средств

- ПК для запуска ПО: ОЗУ объёмом 512 МБ и более, процессор с тактовой частотой 1 ГГц и выше, наличие сетевой карты;

- Веб-камера с минимальным разрешением 1280x720 пикселей и частотой кадров не меньше 24fps.

2.7. Требования к информационной и программной совместимости

2.7.1. Требования к информационным структурам и методам решения

- формирование HSV значений для фильтрации изображения осуществляется на усмотрение пользователя;

- приём и передача программным обеспечением управляющей информации осуществляется с помощью функций;

- вывод информации о направлении взгляда осуществляется по заготовленным алгоритмам.

Чтобы включить распознавание взгляда в свою программу, программисту необходимо подключить необходимые библиотеки по фильтрации и анализу изображения. При каждом новом запуске приложения желательно проводить калибровку, ввиду того, что время суток постоянно меняется, что меняет окружающее освещение, а если веб камера была перемещена на новое место, то калибровка необходима для корректной работы распознавания взгляда.

2.7.2. Требования к программным средствам, используемым программой

Операционная система Windows 7 и выше.

2.8. Требования к безопасности

Секретность информации не предусматривается. Здоровью глаз вред при работе с программой не наносится, так как на них не ведётся никаких излучений. Вред для здоровья глаз может быть нанесен, если установить дополнительное освещение неправильно, из-за чего свет будет направлен прямо в глаза.

3. Архитектура библиотеки типов и основные алгоритмические решения по предлагаемой технологии распознавания направления взгляда.

3.1 Подход к решению проблем распознавания взгляда в задачах программной инженерии

Решение задачи в общем виде выглядит следующим образом:

1) Программа принимает первое изображение в цветовой модели RGB.
2) Пользователь вручную находит пороговые значения для белков и радужек глаза при помощи предоставленного ему графического интерфейса.
3) После того как пользователь нашел значения, они сохраняются и применяются для последующих изображений.
4) При помощи полученных пороговых значений программа производит анализ последующих изображений с целью нахождения и определения направления взгляда.

RGB – аддитивная цветовая модель, описывающая способ кодирования цвета для цветовоспроизведения с помощью трёх цветов: Красного, Зеленого и Синего.

HSV – цветовая модель, в которой координатами цвета являются:

- Hue – цветовой тон. Вариация в пределах 0-360 градусов.
- Saturation – насыщенность. Вариация в пределах 0-100.
- Value – яркость. Вариация в пределах 0-100.

OrigImg – оригинальное изображение, подающееся для первой калибровки Некоторые специальные понятия объяснены по мере их появления в решаемых задачах.

3.2 Решение задачи обнаружения глаз

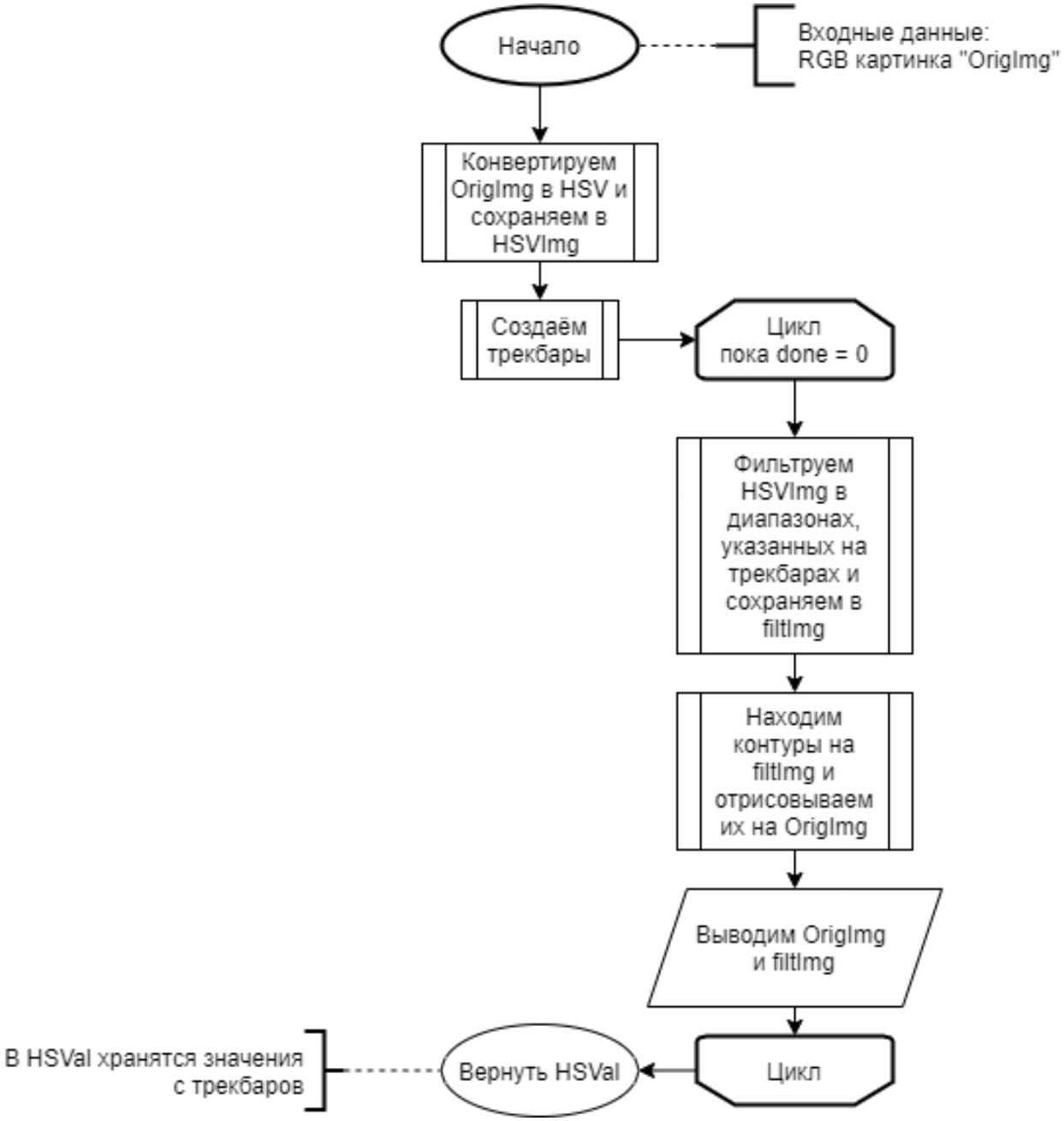


Рис. 1 Функция по нахождению начальных HSV значений

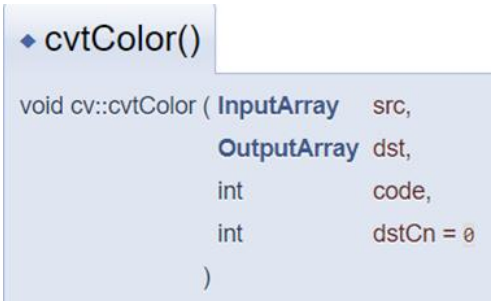
Первая задача состояла в обнаружении белков и радужек глаза на картинке, так как мы полагаемся на цвет картинки, то некоторые универсальные значения применить не получиться из-за непостоянного освещения из-за различного времени суток и различных источников света вокруг.

Для фильтрации и анализа дальнейших изображений, программа будет брать переданную ей первую картинку как основу. Пользователю необходимо вручную отфильтровать картинку для определения диапазонов значений HSV белков и радужек.

Рассмотрим функции этой блок-схемы более подробно:

Функция “Конвертируем OrigImg в HSV и сохраняем в HSVImg”

Этот блок является стандартной функцией OpenCV. Эта функция имеет следующий синтаксис:



src – входное изображение.

dst – изображение, в которое записывается результат.

code – код, уточняющий в какую цветовую модель конвертировать (в нашем случае это COLOR_BGR2HSV).

dstCn – количество каналов в конечном изображении, если параметр равен нулю, то количество каналов определяется автоматически.

Конвертация происходит следующим образом:

$$V \leftarrow max(R,G,B)$$

$$S \leftarrow \begin{cases} \frac{V-min(R,G,B)}{V} & \text{if } V \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$H \leftarrow \begin{cases} 60(G-B)/(V-min(R,G,B)) & \text{if } V = R \\ 120+60(B-R)/(V-min(R,G,B)) & \text{if } V = G \\ 240+60(R-G)/(V-min(R,G,B)) & \text{if } V = B \end{cases}$$

Если H < 0 тогда H ← H + 360.

На выходе 0 <= V <= 1, 0 <= S <= 1, 0 <= H <= 360

Значения потом конвертируются в назначенный тип данных:

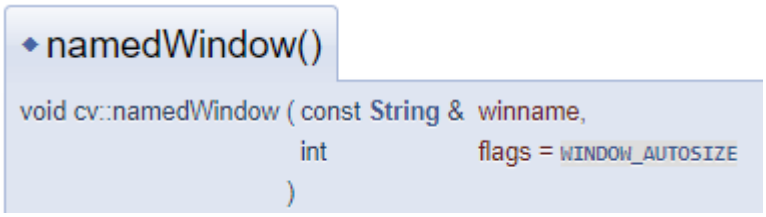
V ←255V, S ← 255S, H ← H/2(для вмещения H в диапазон от 0 до 255)

Функция “Создаём трекбары”

В этой функции создаётся окно и в него помещаются трекбары, при помощи которых пользователь будет манипулировать изображением для фильтрации необходимых ему областей.

Создание окон и трекбаров осуществляется при помощи стандартных функции OpenCV, функции имеют следующий синтаксис:

Функция для создания окон:



winname – имя окна, которое будет отображаться, также используется как идентификатор окна,

flags – флаги окна, в основном служат для ркгулировки размера окна при его создании.

Функция для создания трекбаров:

◆ createTrackbar()

```
int cv::createTrackbar ( const String &     trackbarname,
                         const String &     winname,
                         int *              value,
                         int                count,
                         TrackbarCallback   onChange = 0,
                         void *             userdata = 0
                       )
```

trackbarname – имя созданного трекбара.

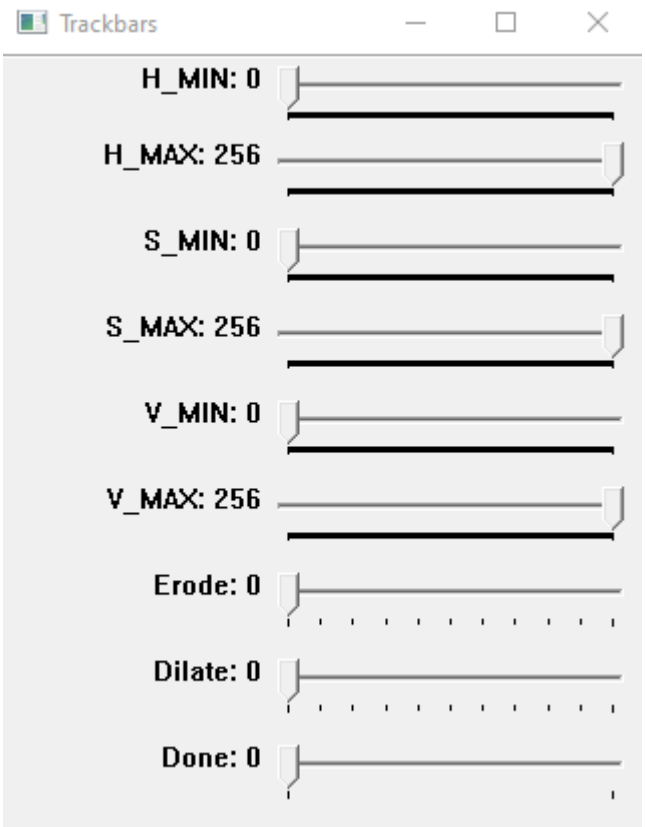
winname – имя окна на котором будет нарисован трекбар.

value – указатель на целочисленную переменную, чье значение отображает положение слайдера.

count – максимальная позиция слайдера. (минимальная всегда равна 0)

onChange – указатель на функцию, которая будет вызываться при изменении положения слайдера.

userdata – указатель на переменную, которая передаётся в функцию onChange.



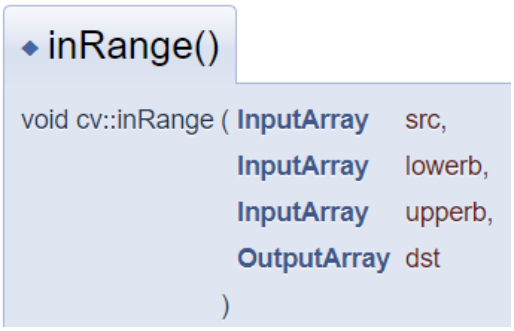
В результате выполнения этой функции выводится следующее окно:

Трекбар – это полоска, которая имеет ползунок, движение которого меняет значение, которое было привязано к нему программистом. Движение ползунка осуществляется при помощи нажатия на него левой кнопки мыши и перетаскиванием влево/вправо. Количество доступных значений на

трекбаре ограничивается программистом. В нашем случае к трекбарам привязаны HSV значения для фильтрации.

Функция “Фильтруем HSVImg в диапазонах, указанных на трекбарах и сохраняем в filtImg”

Фильтрация проходит при помощи функции OpenCV, на которую подаются значения с окна с трекбарами, синтаксис функции выглядит следующим образом:



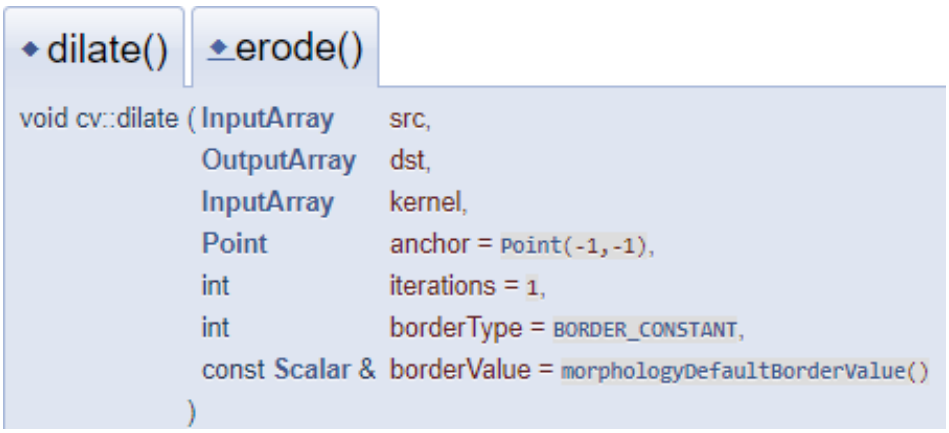
src – массив**,** содержащий изображение, которое будем фильтровать.
lowerb – нижняя граница массива.
upperb – верхняя граница массива.
(так как у нас 3х канальное изображение, мы используем скаляр для верхней и нижней границы)
dst – массив в который будет записано модифицированное изображение.

Описание работы функции:
Функция проверяет диапазон следующим образом

$$\texttt{dst}(I) = \texttt{lowerb}(I)_0 \leq \texttt{src}(I)_0 \leq \texttt{upperb}(I)_0 \land \texttt{lowerb}(I)_1 \leq \texttt{src}(I)_1 \leq \texttt{upperb}(I)_1 \land \\ \land \texttt{lowerb}(I)_2 \leq \texttt{src}(I)_2 \leq \texttt{upperb}(I)_2$$

То есть для dst(I) установлено значение 255 (все 1-биты), если src(I) находится в пределах указанного поля 1D, 2D, 3D, ..., и 0 в противном случае.

Также помимо фильтрации по значением были добавлены функции **erode** и **dilate**, которые работают следующим образом:



src – матрица с входным изображением.

dst – матрица, куда будет записан результат, должна быть того же размера, что и src.

kernel – элемент, используемый для эрозии(erode) и расширения(dilate) (например квадрат, размером 3 на 3 пикселя).

anchor – координаты внутри элемента, относительно которых элемент будет нарисован.

iterations – сколько раз нужно применить операцию.

borderType – метод экстраполяции пикселей.

borderValue – величина границ, если borderType = BORDER_CONSTANT.

Функция расширяет(сужает) исходное изображение с помощью указанного элемента структурирования, который определяет форму окрестности пикселя, по которой берется максимум(минимум):

$$\texttt{dst}(x,y) = \max_{(x',y'):\, \texttt{element}(x',y')\neq 0} (\min) \; \texttt{src}(x+x', y+y')$$

Функция “Находим контуры на filtImg и отрисовываем их на OrigImg”

Нахождение контуров и их отрисовка на оригинальном изображении происходит при помощи следующих функций OpenCV:

Функция для нахождения контуров:

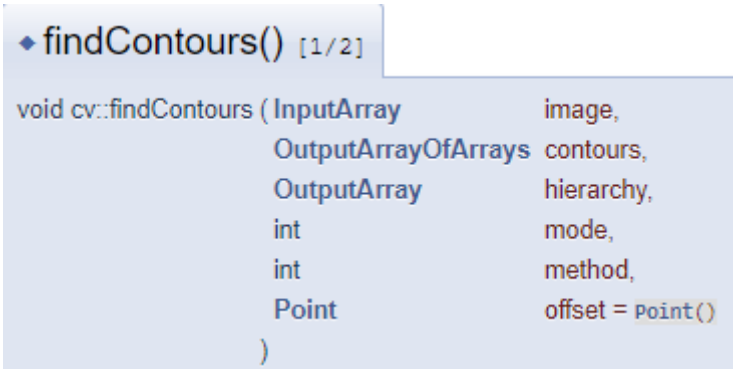


image – входное изображение, все ненулевые пиксели считаются за 1, все нулевые остаются 0

contours – обнаруженные контуры

hierarchy – вектор топологии изображения. В данном векторе столько же элементов, сколько контуров.

mode – режим получения контуров

method – метод апроксимации контуров

offset – смещение каждой точки контуров

Функция отрисовки контуров:

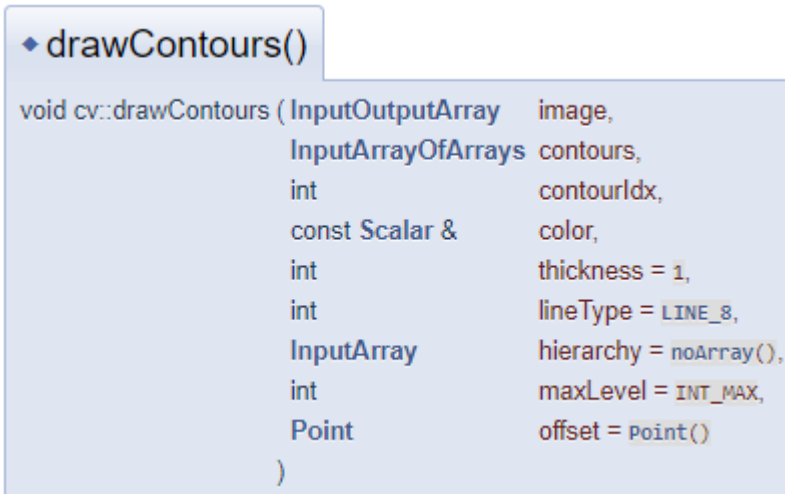


image – Изображение на котором производится отрисовка

contours – Все найденные для отрисовки. Каждый контур представляет из себя массив точек.

contourIdx – целочисленный параметр, показывающий, какой контур нужно отобразить. Если число отрицательное, то отрисовываются все контуры, которые были переданы как аргумент contours

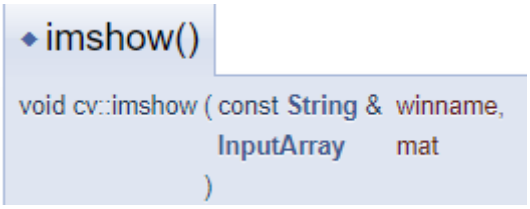
color – цвет контуров

thickness – толщина линий для отрисовки

Остальные параметры использованы не были

Блок “Выводим OrigImg и filtImg”

Вывод изображений производится при помощи следующей функций:



winname – имя окна, в котором будет выведено изображение

mat – массив, в котором находится изображение

Блок “вернуть HSVal”

В конечном счёте функция возвращает переменную HSVal, в которой находятся значения трекбаров. Переменная HSVal представляет из себя целочисленный массив с 8 элементами, для удобства обращения к элементам которого было создано перечисление:
элемент 0: HMIN – нижний порог фильтрации для цветового тона
элемент 1: SMIN – нижний порог фильтрации для насыщенности
элемент 2: VMIN – нижний порог фильтрации для яркости
элемент 3: HMAX – верхний порог фильтрации для цветового тона
элемент 4: SMAX – верхний порог фильтрации для насыщенности
элемент 5: VMAX – верхний порог фильтрации для яркости
элемент 6: ERODECOUNT – сколько раз нужно применить функцию erode
элемент 7: DILATECOUNT – сколько раз нужно применить функцию dilate

Действия для проведения фильтрации.

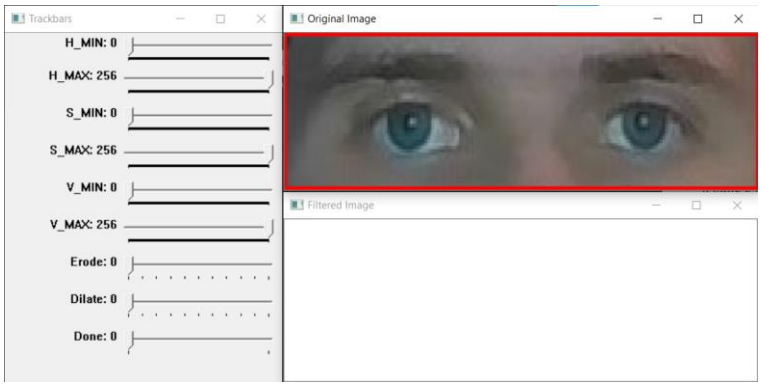
Для того чтобы провести фильтрацию, необходимо двигать ползунки на трекбарах для подборки значений фильтрации. Изменение ползунков приводит к следующим изменениям:
- в окне с названием “Original Image” отмечаются найденные контуры с поставленными знчениями. Нужно быть внимательным, тк если на OriginalImage есть большой квадратный контур по периметру всего

изображения, то следует продолжить фильтрацию, для его исключения, тк он помешает дальейшей работе программы

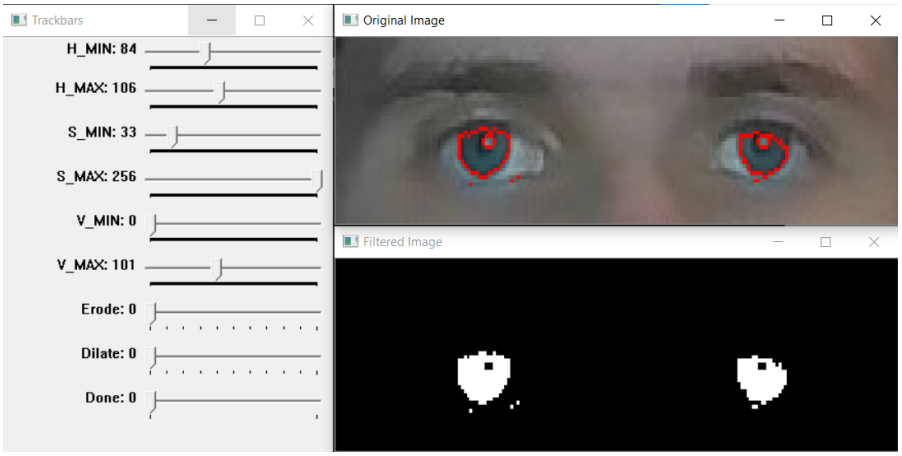
- в окне с названием “Filtered Image” чёрным будут отмечатся те цвета, от которых программа избавилась, а белым цветом будут отмечаться те цвета, которые принимаются для дальнейшей работы программы

До фильтрации:

Здесь можно увидеть красный контур по периметру изображения, если он присутствует, значит либо нужно фильтровать дальше, либо фильтрация проведена неверно и нужно поменять значения фильтрации или попробовать вернуть ползунки в исходное положение и попробовать снова



После фильтрации:
Здесь был успешно отфильтрован и найден зрачек. Шумы будут проигнорированы программой.



3.3 Решение задачи по определению направления взгляда

После получения необходимых значений для фильтрации, можно приступать к анализу следующих изображений, сделанных при тех же условиях, что и на первом изображении. Второй основной задачей стояло непосредственно обнаружение направления взгляда, блок схема функции по обнаружению взгляда представлена ниже:

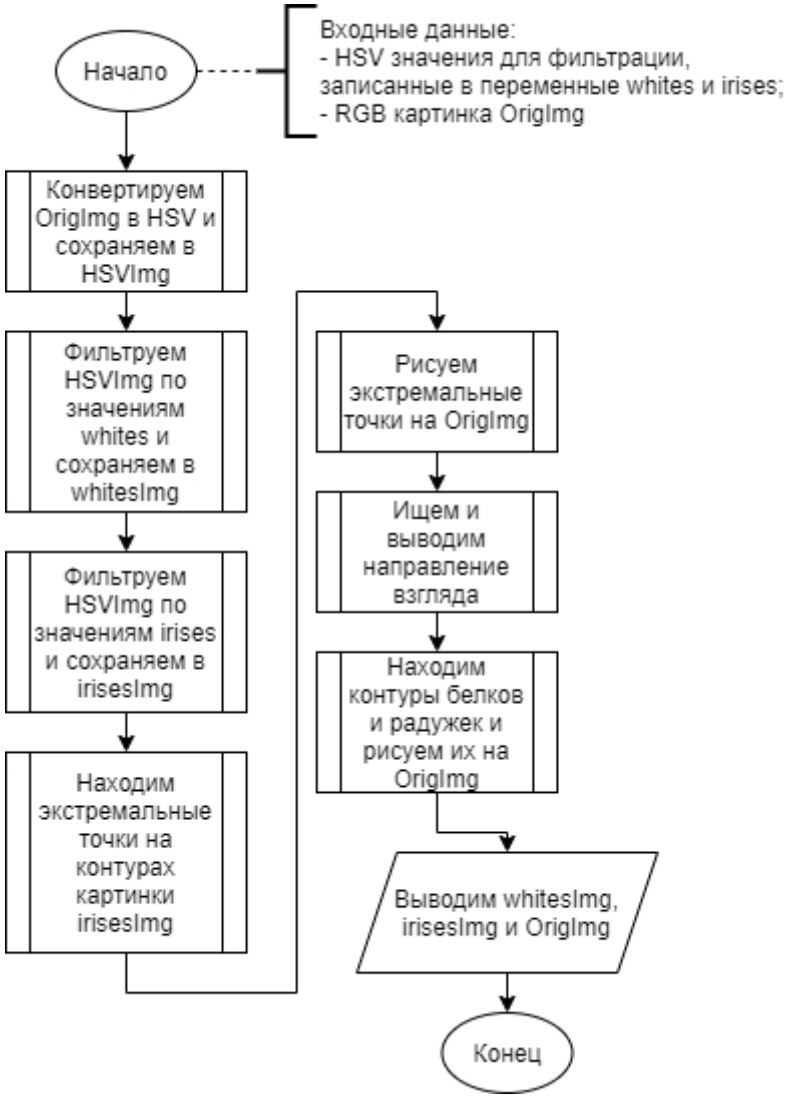


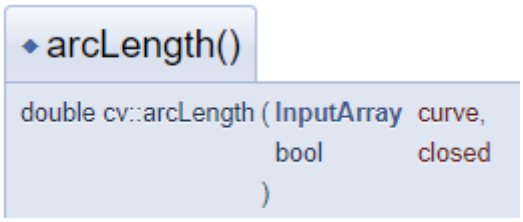
Рис. 2 Функция по определению направления взгляда

Далее функции на этой блок-схеме рассматриваются более подробно:

Функция “Конвертируем OrigImg в HSV и сохраняем в HSVImg” уже была рассмотрена в первой задаче, они полностью идентична функции с таким же названием, а функции **“Фильтруем HSVImg по значениям whites и сохраняем в whitesImg”** и **“Фильтруем HSVImg по значениям irises и сохраняем в irisesImg”** полностью аналогичны функции **“Фильтруем HSVImg в диапазонах, указанных на трекбарах и сохраняем в filtImg”**

Функция “Находим экстремальные точки на контурах картинки irisesImg”

Эта функция первым делом определяет, какой из контуров имеет больший периметр; сделано это для того, чтобы предотвратить поиск экстремальных точек на границах лишних областей, которые не могли быть отфильтрованы(шумы). Поиск осуществлялся при помощи функции **arcLength**, которая имеет следующий синтаксис:



curve – вектор точек в 2d пространстве

closed – флаг, который даёт понять функции, замкнут ли контур или нет.

После определения контура с большим периметром, определяются 4 экстремальные точки на этом контуре, путём поиска точек с максимальными и минимальными значениями координат x и y. Блок схема работы поиска приведена ниже.

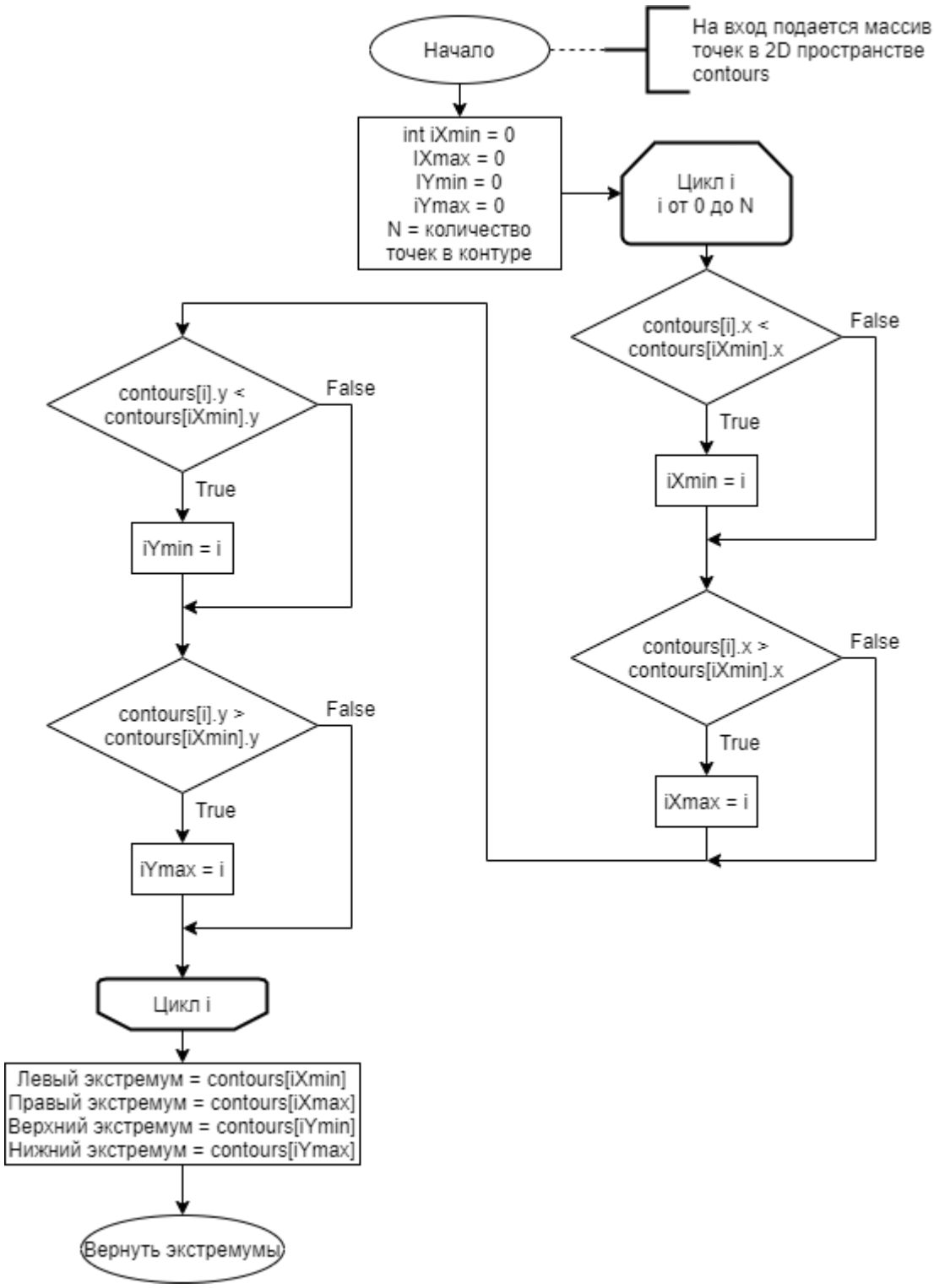


Рис. 3 Алгоритм поиска экстремумов на контуре

Функция “Рисуем экстремальные точки на OrigImg”

Простая функция, которая берёт координаты найденных экстремумов и рисует на этих координатах небольшие окружности, осуществляется это при помощи стандартных функций OpenCV:

◆ Circle()

```
cv::gapi::wip::draw::Circle::Circle ( const cv::Point &   center_,
                                      int                 radius_,
                                      const cv::Scalar &  color_,
                                      int                 thick_ = 1,
                                      int                 lt_ = cv::LINE_8,
                                      int                 shift_ = 0
                                    )
```

center_ - центр окружности

radius_ - радиус окружности

color_ - цвет окружности

thick_ - толщина окружности

◆ Line()

```
cv::gapi::wip::draw::Line::Line ( const cv::Point &   pt1_,
                                  const cv::Point &   pt2_,
                                  const cv::Scalar &  color_,
                                  int                 thick_ = 1,
                                  int                 lt_ = cv::LINE_8,
                                  int                 shift_ = 0
                                )
```

pt1_ - Точка 1

pt2_ - Точка 2

color_ - Цвет

thick_ - толщина линии

Функция “Ищем и выводим направление взгляда”

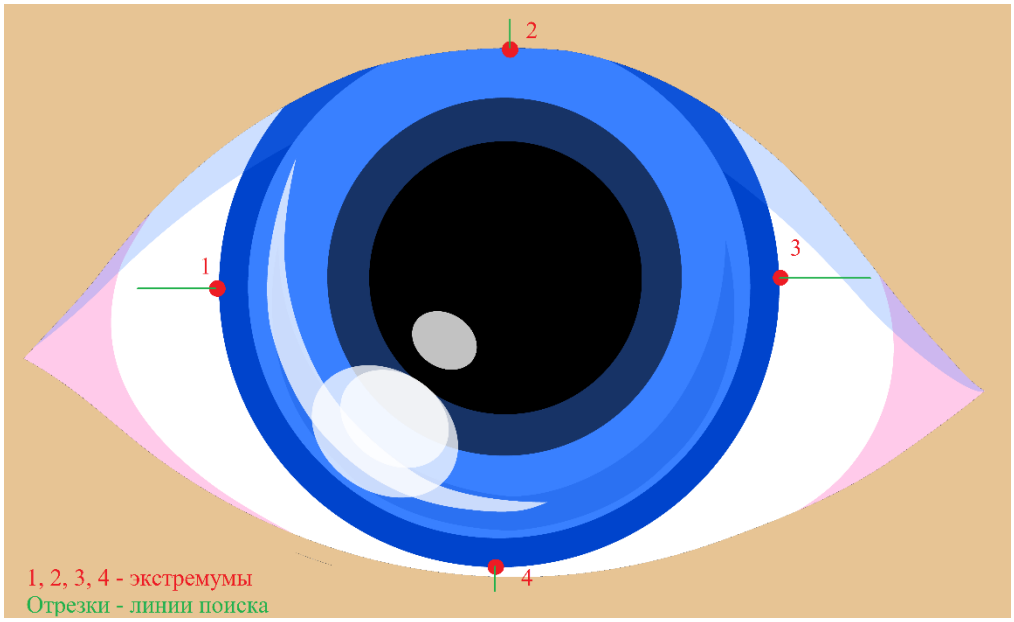


Рис. 4 Предположительная работа алгоритма

При помощи найденных экстремальных точек и отфильтрованных изображений, программа должна быть в состоянии определить, в какую сторону направлен взгляд. В зависимости от того, обнаружены ли белки относительно экстремумов.

4. Состав и организация тестовых примеров по демонстрации предлагаемой библиотеки типов.

Для тестирования программы сделаны группы фотографий с различным направлением взгляда. У каждой группы будут различающиеся обстоятельства. Обстоятельствами для изменения являются:

- уровень освещенности помещения: без дополнительного освещения/с дополнительным освещением;
- положение лица относительно веб-камеры: на расстоянии/приближенно к камере;
- веб-камера: камера ноутбука/камера смартфона.

Начнем с самых плохих условий:

Тест 1. Снимки с веб-камеры ноутбука на расстоянии без дополнительного освещения

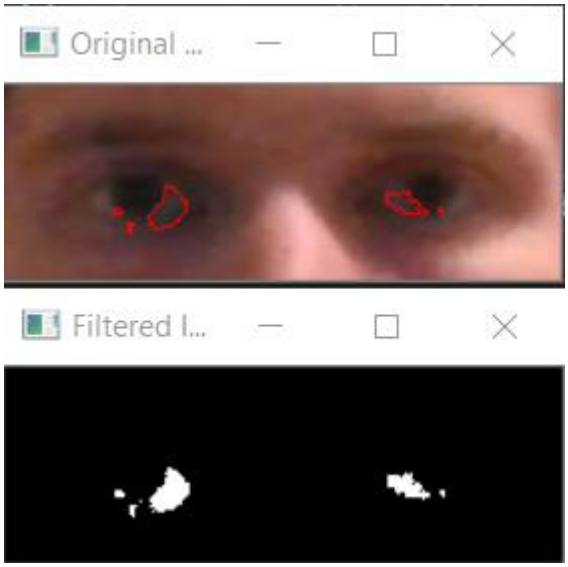


Рис.5 Фильтрация белков

Этот тест оказался неудачным на этапе фильтрации первого изображения для дальнейшего анализа других изображений из этой же группы. Наиболее приемлемый результат в фильтрации белков представлен на рисунке 5. Хотя оказалось возможным найти хоть какую-то область белков

на картинке, этой области недостаточно, для того, чтобы точно оценить направление взгляда.

Тест 2. Снимки с веб камеры ноутбука на расстоянии с дополнительным освещением

Рис. 6 Обнаружение взгляда прямо

Дополнительное освещение позволило немного улучшить результат, на рисунке 6 зелёным цветом обводятся радужки, а синим цветом обводятся белки, программа так же вывела, что взгляд направлен вперёд:

```
Eye's direction is Somewhere (probably an error) forward (eye is closed slightly)
```

Но это больше похоже на удачное стечение обстоятельств, чем на чёткую работу алгоритма.

Теперь можно попробовать с этими значениями для фильтрации найти направления взгляда на других изображениях в этой же группе:

Рис. 7 Обнаружение взгляда влево

На удивление, алгоритм по фильтрации сработал даже лучше, чем на первичном изображении. Но определение направления всё же оказалось неверным: `Eye's direction is Somewhere (probably an error) up`

Тест 3. Снимки с веб камеры ноутбука вблизи без дополнительного освещения.

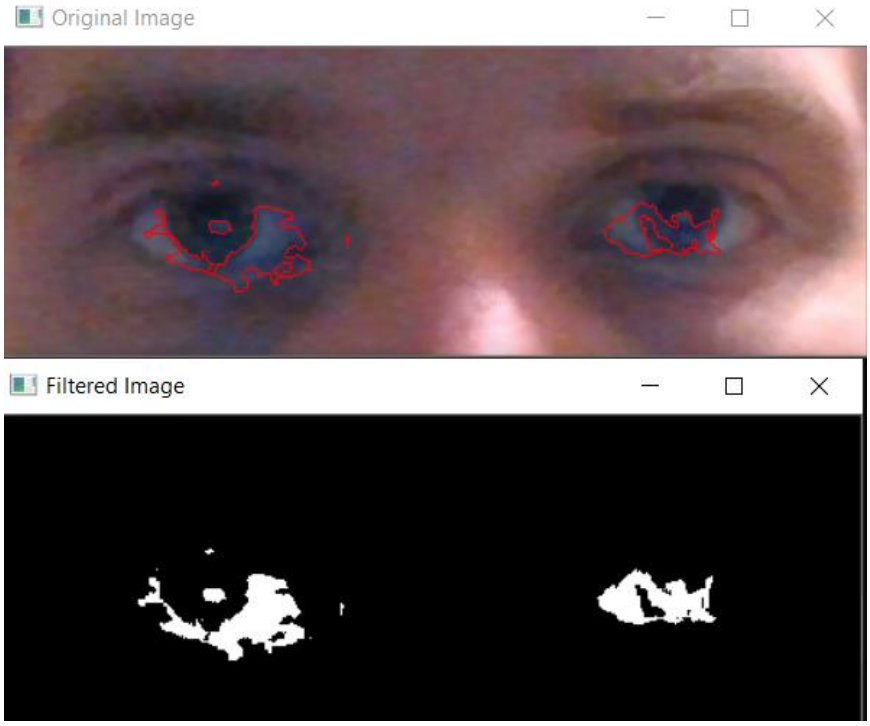


Рис. 8 Фильтрация белков

Приближение лица к камере даёт большее количество информации для обработки, так как пикселей становиться больше, но из этого теста можно увидеть, что простое приближение к камере хоть и дало более чёткую картину, фильтрация происходит недостаточно хорошо.

Тест 4. Снимки с веб камеры ноутбука вблизи с дополнительным освещением.

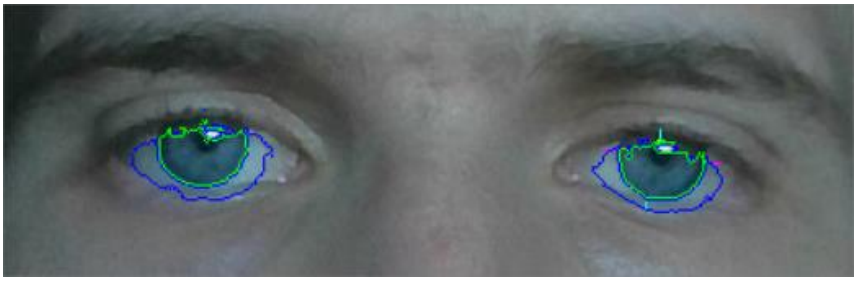


Рис. 9 Обнаружение взгляда вперёд

На данном изображении, как видно из рисунка 9, фильтрация прошла очень хорошо, увеличение количества пикселей за счёт приближения к камере и хорошее освещение позволяют производить достаточно точную фильтрацию для обработки информации о направлении взгляда.

Посмотрим, что будет, если применить эти же данные для фильтрации к другой картинке из этой же группы:

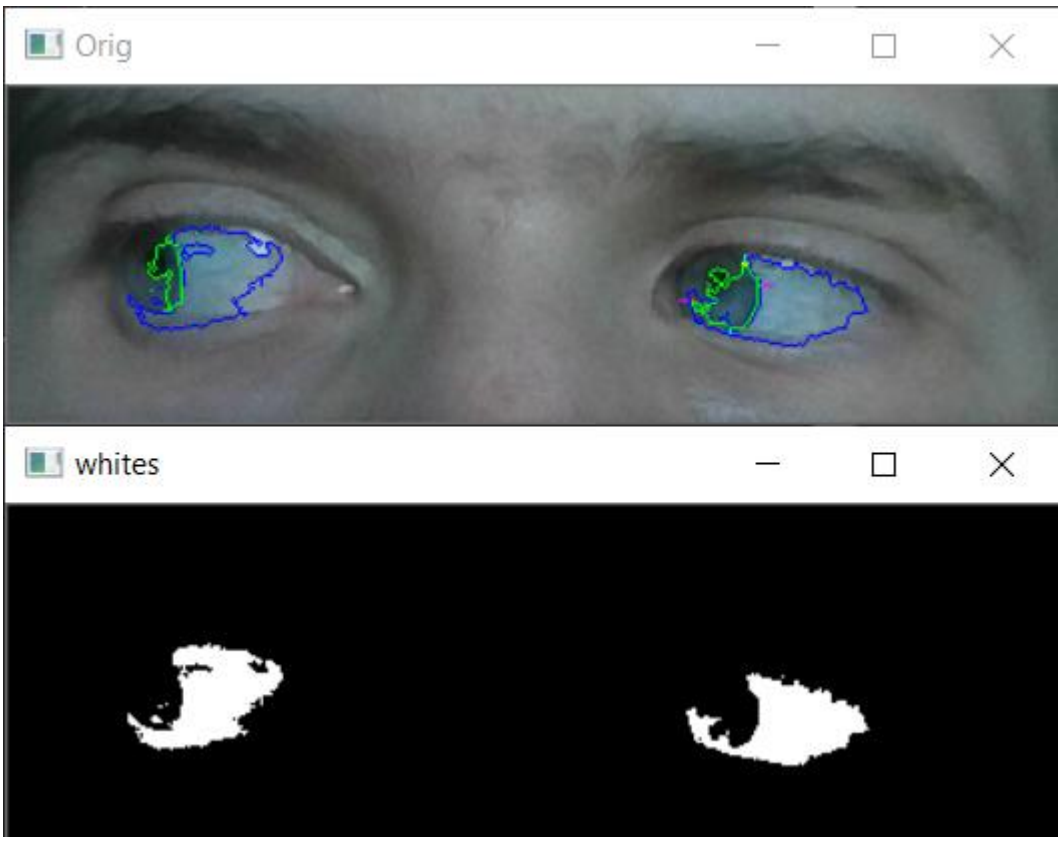


Рис. 10 Обнаружение взгляда влево

Как видно из Рисунка 10 фильтрация белков сработала нормально, но фильтрация радужек оставляет желать лучшего. Хотя мы и улучшили все условия для фильтрации изображения, фильтрация прошла не совсем корректно. Возможно изменение веб-камеры способно решить проблему.

Тест 5. Снимки с веб камеры смартфона на расстоянии без дополнительного освещения.

Попытка применить другую веб-камеру для проводимых тестов.

Рис. 11 Обнаружение взгляда прямо

Увеличение количества обработки информации позволило увеличить точность нахождения контуров, они теперь более явные, чем во всех предыдущих тестах. Так же добавилась дополнительная сложность при начальной фильтрации:

1) Увеличение количества пикселей на изображении привело к тому, что на картинке теперь больший простор для пикселей различного цвета, что означает дополнительные лишние контуры при фильтрации.
2) Так же начальная фильтрация стала намного более строгой, из-за чего приходится тратить больше времени на первоначальную фильтрацию белков и радужек.

Тест 6. Снимки с веб камеры смартфона на расстоянии с дополнительным освещением.

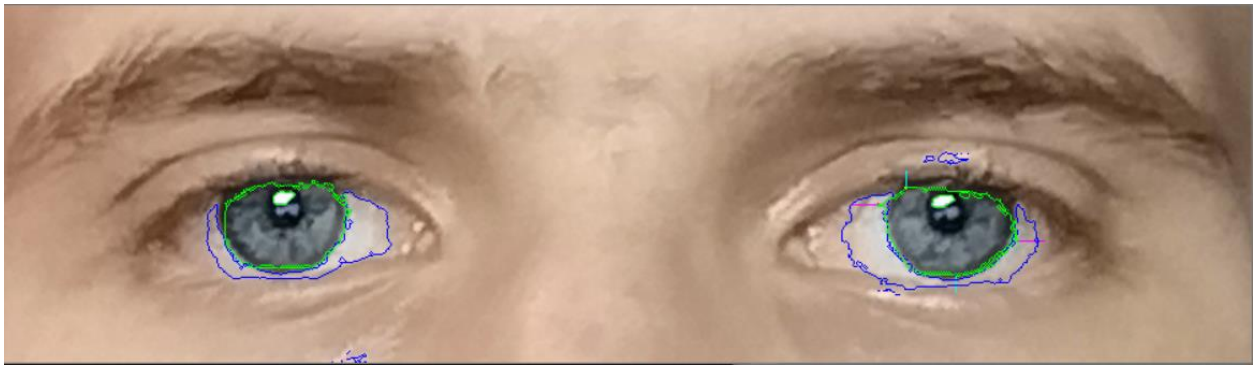


Рис. 12 Обнаружение взгляда вперёд

Дополнительное освещение дало о себе знать, фильтрация прошла проще и на изображении намного меньше лишних контуров от шумов. Далее рассматривается фильтрация других изображений с теми же значениями для фильтрации, что и для первичного изображения:

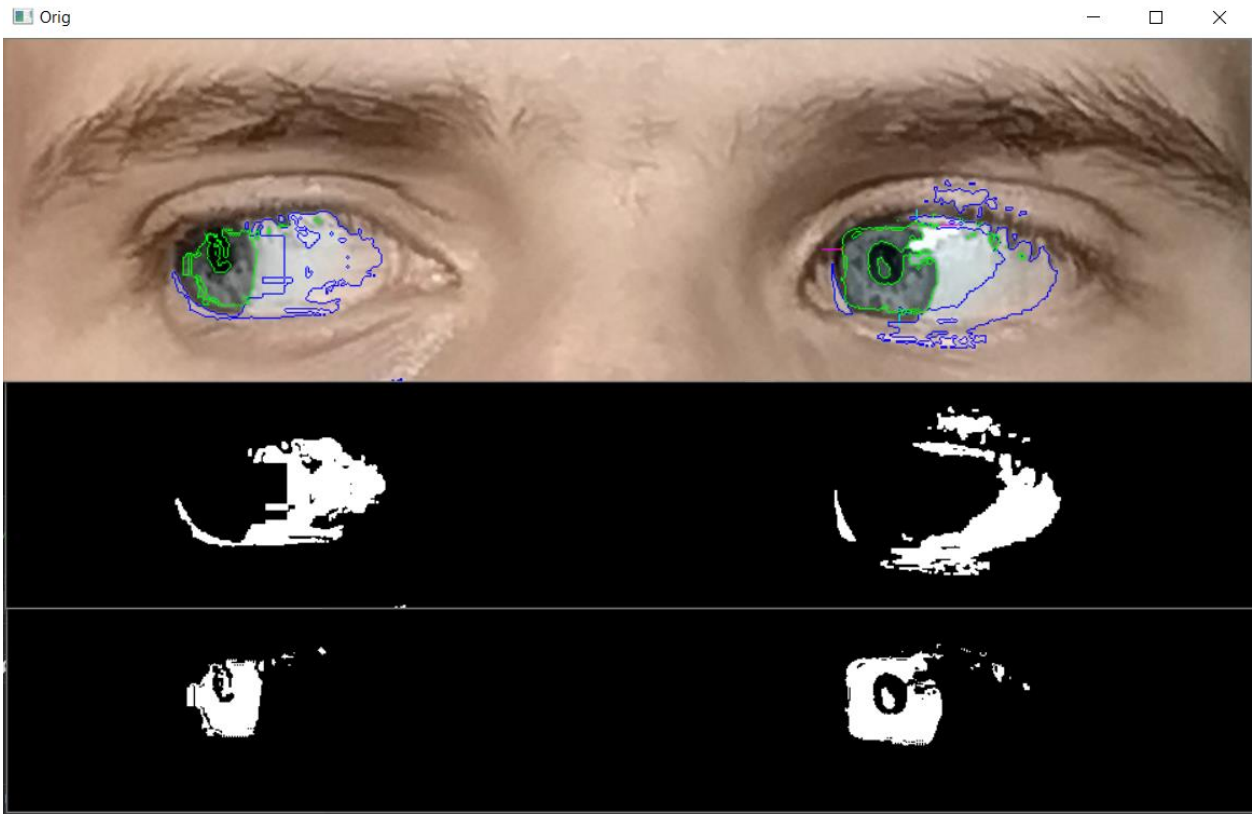


Рис. 13 Обнаружение взгляда влево

Теперь наблюдается ситуация, обратная Тесту 4. Фильтрация радужек прошла достаточно хорошо, в то время как белки были отфильтрованы плохо.

Тест 7. Снимки с веб камеры смартфона вблизи без дополнительного освещения.

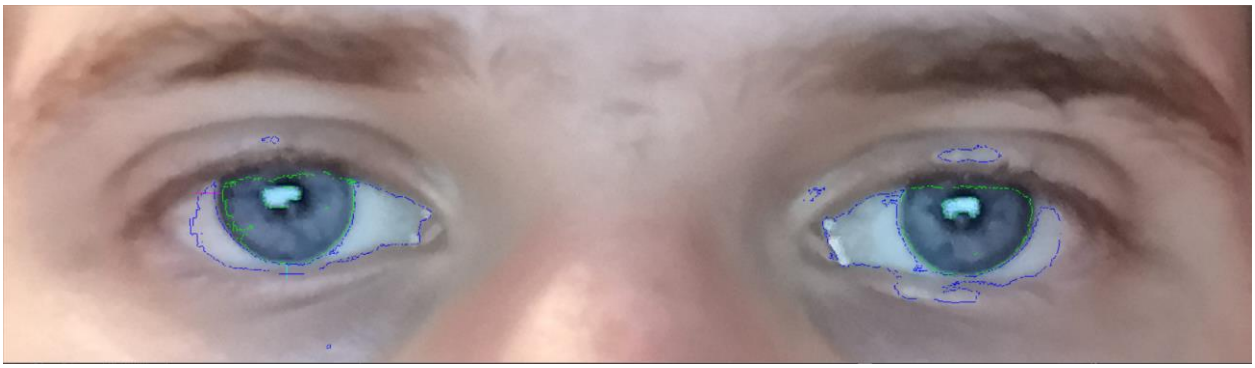


Рис.14 Обнаружение взгляда прямо

Снимок на хорошую камеру, даже без дополнительного освещения дал исключительный результат, контуры почти идеально показывают, где находятся белки и радужки. Применим значения фильтрации первоначального изображения на другое изображение из этой группы:

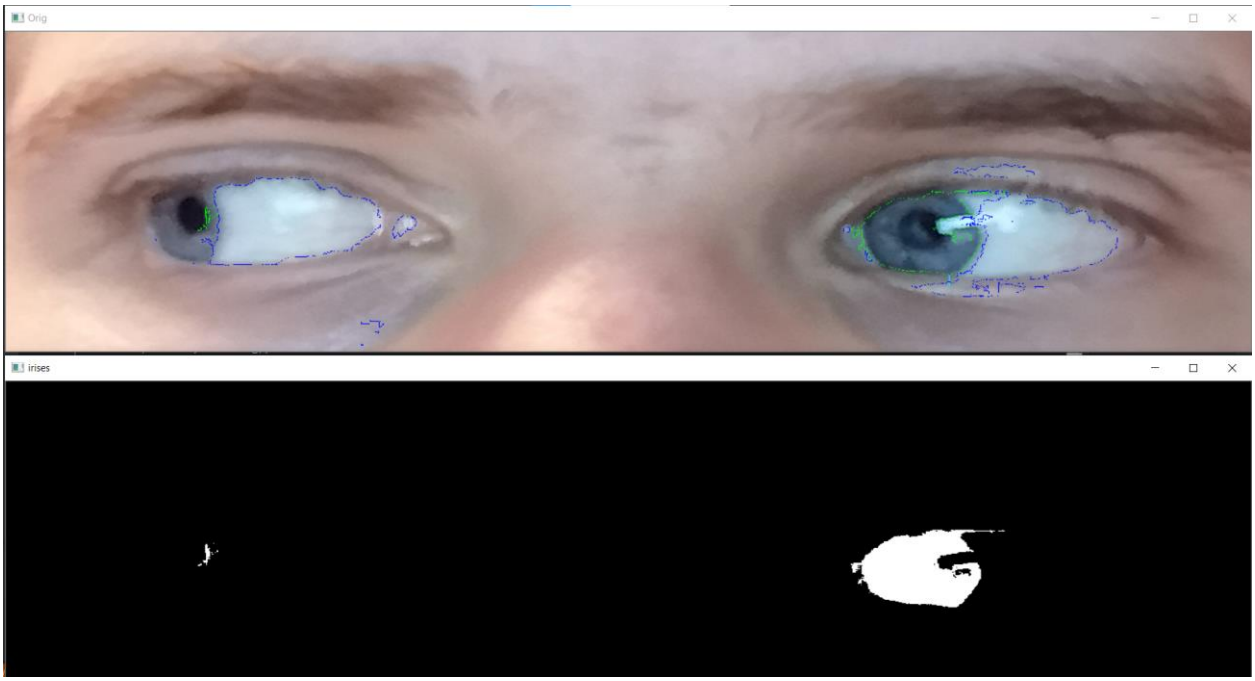


Рис. 15 Обнаружение взгляда влево

И снова имеем ситуацию, в которой фильтрация прошла не так, как надо, но данный случай несколько уникален. Хотя белки и были найдены корректно, радужка, по неизвестной причине, была найдена только одна.

Тест 8. Снимки с веб камеры смартфона вблизи с дополнительным освещением.

Рис 16. Обнаружение взгляда прямо

Это самый лучший результат, среди всех тестов, что и ожидаемо. Огромное количество пикселей для обработки и хорошее освещение позволяет намного проще фильтровать необходимые контуры. Теперь проверим значения фильтрации первоначального изображения на другом изображении из этой группы.

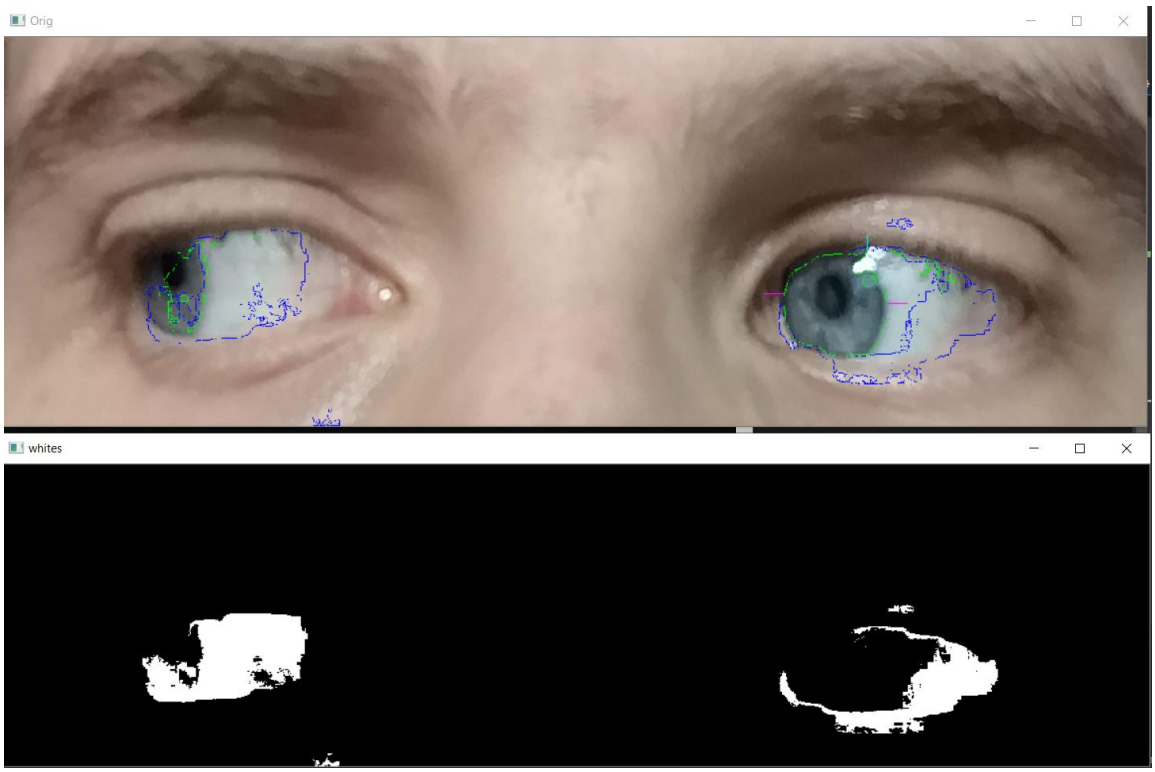


Рис. 17. Обнаружение взгляда влево

Несмотря на все положительные условия, которые были установлены для улучшения точности нахождения контуров, фильтрация всё ешё происходит не успешно, в данном тесте ни белки, ни радужки не были корректно найдены.

Тест 9, 10 Попытка использования разработанного программного комплекса в условиях обыкновенной аудитории.

Этот тест был произведён в более обыденных условиях, в случае если нужно спонтанно воспользоваться, а необходимых условий для программы нет

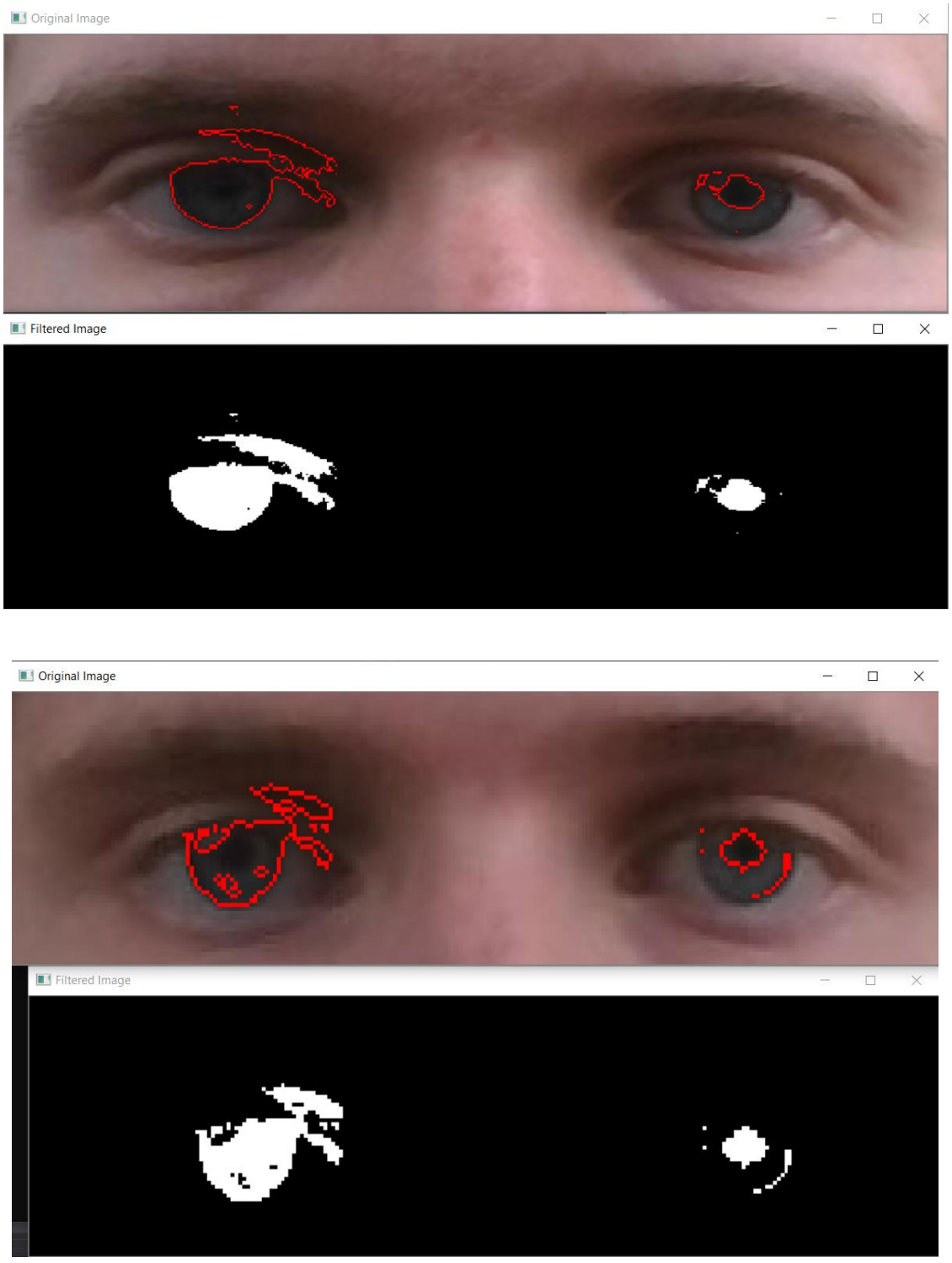


Рис.18 Обнаружение глаз прямо

Данный тест показывает, что обыденное использование этой программы возможна, если только все условия для хорошего отслеживания выполнены без оборудования.

Тест 11 Применение программы на красных глазах.
В этом тесте была взято изображение из интернета с красными глазами и проведён тест по обнаружению белков и радужек на изображении.

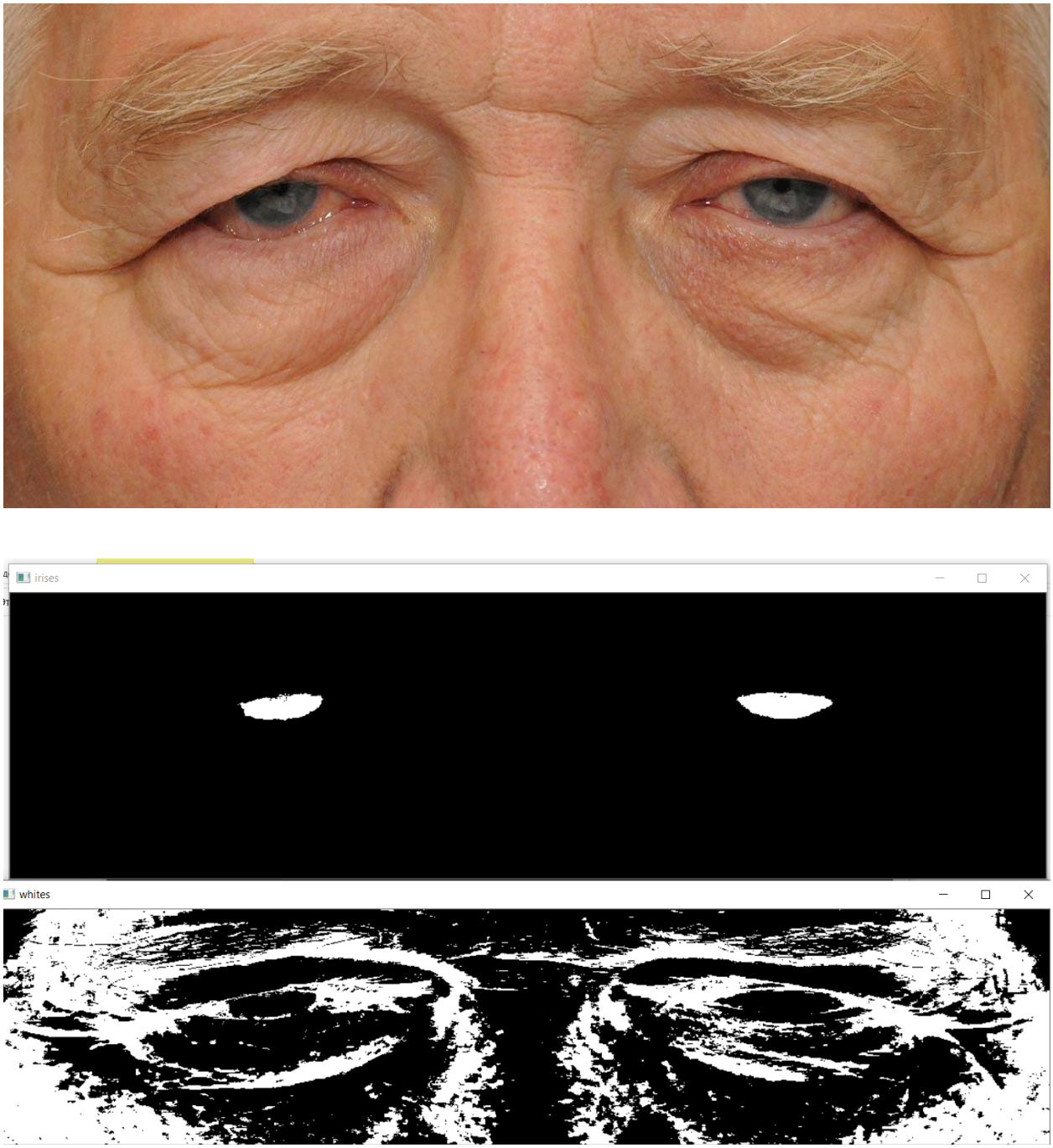


Рис. 19 Фильтрация “красных” глаз

На таком изображении с высоким разрешением радужки нашлись легко, но из-за покрасневших глаз, белки и кожа лица были приблизительного цвета из за чего фильтрация белков оказалась невозможной.

Тест 12 Тест на высококачественном изображении

В данном тесте использовано изображение с разрешением 4144x2480 пикселей. На изображении видно, что глаз освещен неравномерно, с лево стороны глаза отражается окно, а правая часть глаза тёмная.

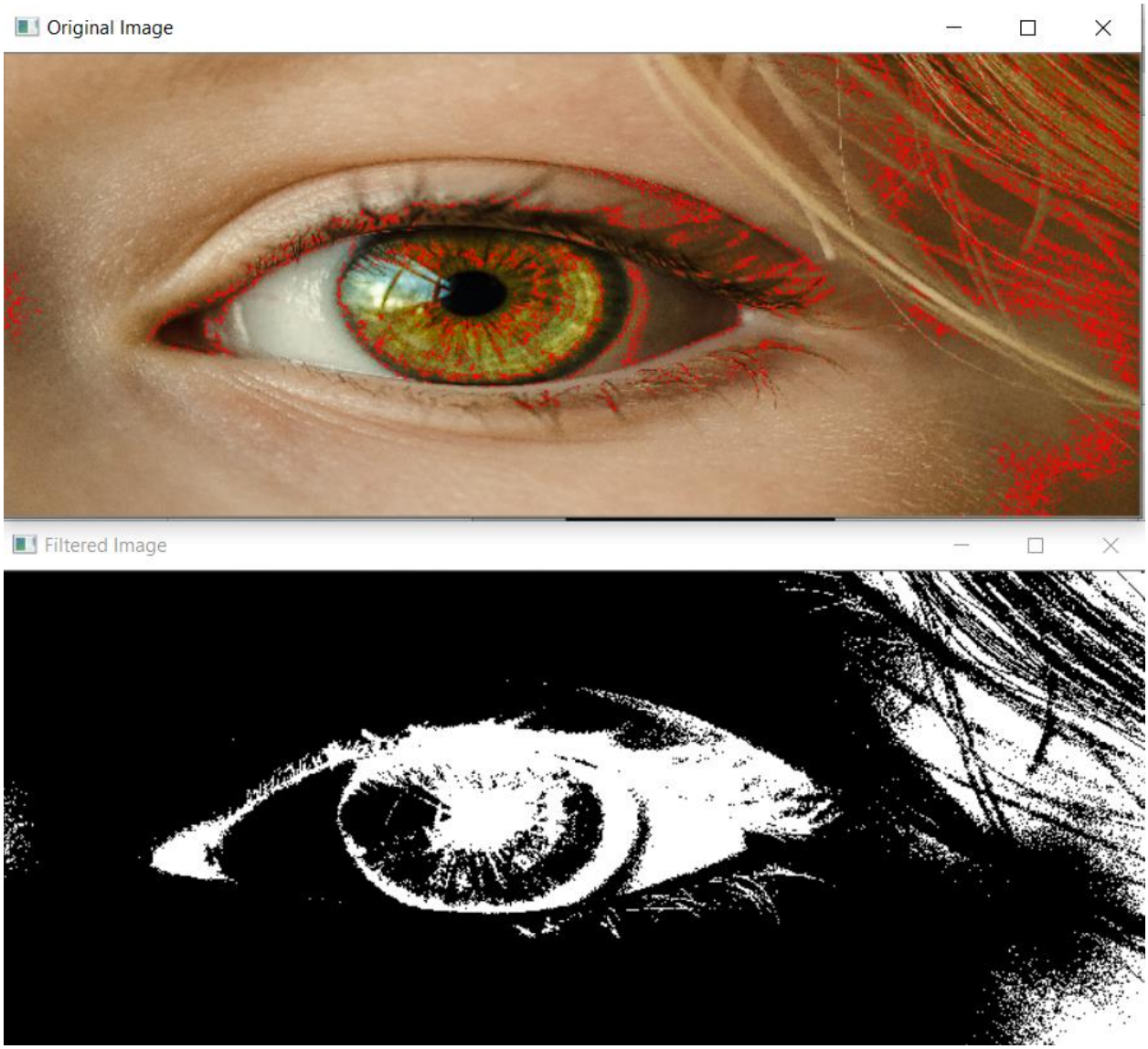


Рис. 20 Фильтрация высококачественного изображения

№ Теста	Состояния			Первичная фильтрация		Фильтрация след. избр.	
	К	**П**	**С**	Белков	Радужек	Белков	Радужек
Тест 1 (1280x720)	0	0	0	✗ 134/286	✗ 270/374	✗ x/x	✗ x/x
Тест 2 (1280x720)	0	0	1	✓ 159/264	✓ 166/215	✓ 163/220	✓ 95/112
Тест 3 (1280x720)	0	1	0	✗ x/x	✗ x/x	✗ x/x	✗ x/x
Тест 4 (1280x720)	0	1	1	✓ 686/840	✓ 745/964	✓ 560/968	✗ 250/810
Тест 5 (3456x4608)	1	0	0	✓ 5294/5432	✗ 6141/7213	✗ 2563/8140	✗ 5841/5390
Тест 6 (3456x4608)	1	0	1	✓ 2991/3245	✓ 4792/5580	✗ 2763/6976	✓ 3667/4156
Тест 7 (3456x4608)	1	1	0	✓ 24887/26062	✓ 36390/38654	✓ 40689/43309	✗ 33913/38635

5. Результаты тестирования.

Тест 8 (3456x4608)	1	1	1	✓ 19225/22687	✓ 31912/33641	✗ 15843/40587	✗ 24664/35789
Тест 9 (1280x720)	0	0	0	✗ x/x	✗ x/x	✗ x/x	✗ x/x
Тест 10 (1280x720)	0	1	0	✗ x/x	✗ x/x	✗ x/x	✗ x/x
Тест 11 (1200x577)	0	1	0	✗ x/x	✓ 2874/3278	-	-
Тест 12 (4144x2480)	1	1	0	✗ x/x	✗ x/x	-	-

Результаты тестирования показывают, что разработанный алгоритм оказался не самым эффективным, а также имеются некоторые ограничения для использования программы. В таблице ниже в колонке “Состояния” сокращения расшифровываются следующим образом:

К – камера, показывает какая камера была использована
(0 – камера ноутбука; 1 – камера смартфона)

П – Положение лица относительно веб камеры
(0 – лицо на расстоянии, 1 – лицо вблизи)

С – Наличие дополнительного освещения
(0 – отсутствие дополнительного освещения, 1 – наличие дополнительного освещения)

Соотношения чисел в колонках “Белков” и “Радужек” – это соотношение найденных пикселей белков/радужек к реальному числу пиксей белков/радужек**.**

Из представленных выше результатов можно сделать следующие выводы:

1) Наличие дополнительного освещения гарантирует хорошую фильтрацию белков и радужек

2) Улучшение качества камеры не обязательно будет гарантировать лучшее качество фильтрации
3) Приближение и отдаление от веб-камеры не даёт каких-то существенных результатов

Заключение

В данной диссертации выполнены следующие задачи:
Проведено исследование на темы “Отслеживание глаз” и “Отслеживание взгляда”.
Изучены различные подходы и алгоритмы для отслеживания взгляда.
Рассмотрены различные перспективные области применения для использования отслеживания глаз и взгляда.
Разработан алгоритм для отслеживания направления взгляда при помощи считанной информации с веб-камеры.
Предложен алгоритм работы программы по отслеживанию взгляда
Разработана программа, реализующая предложенный алгоритм.
Проведены тесты отслеживания направления взгляда с променением разработанной программы и сделаны выводы, основанные на этих тестах.

Проделанная работа имеет частично положительный результат. Написанная программа позволяет выделять контуры белков и радужек при определенных благоприятных условиях. Но алгоритм для анализа информации о полученных контурах работает с большими неточностями. Получившийся программный комплекс требует доработки, а именно в функции анализа полученной информации. Применение алгоритма для анализа площади белков глаз мог бы значительно улучшить определение направление взгляда.

Библиография

1. Huey, E.B. (1968). The psychology and pedagogy of reading. Cambridge, MA: MIT Press.
2. Buswell, G.T. (1922). Fundamental reading habits: A study of their development.
3. Buswell G.T. (1935). How People Look at Pictures.
4. Yarbus, A. L. Eye Movements and Vision.
5. Rayner, K. (1978). Eye movements in reading and information processing.
6. Just MA, Carpenter PA (1980) A theory of reading: from eye fixation to comprehension.
7. Posner, M. I. (1980) Orienting of attention.
8. Wright, R.D., & Ward, L.M. (2008). Orienting of Attention.
9. Hoffman, J. E. (1998). Visual attention and eye movements. In H. Pashler.
10. Deubel, H. & Schneider, W.X. (1996) Saccade target selection and object recognition: Evidence for a common attentional mechanism.
11. Holsanova, J. (2007) Picture viewing and picture descriptions.
12. IndustryArc (2019) Eye Tracking Technology: Applications & Future Scope.
13. Inc (2019) Читаем по глазам: как digital-анализ человеческого взгляда меняет жизнь
14. Bulling, A. Robust Recognition of Reading Activity in Transit Using Wearable Electrooculography
15. Bulling, A. et al.: Eye Movement Analysis for Activity Recognition
16. Bulling, A. et al.: Eye Movement Analysis for Activity Recognition Using Electrooculography
17. Bulling, A.; Roggen, D. and Tröster, G. Wearable EOG goggles: Seamless sensing and context-awareness in everyday environments
18. Wikipedia. Окулография

19. Elbert, T., Lutzenberger, W., Rockstroh, B., Birbaumer, N., 1985. Removal of ocular artifacts from the EEG. A biophysical approach to the EOG. Electroencephalogr Clin Neurophysiol
20. Keren, A.S., Yuval-Greenberg, S., Deouell, L.Y., 2010. Saccadic spike potentials in gamma-band EEG: Characterization, detection and suppression.

Приложение 1. Функция main

```
#include <iostream>
#include <string>
#include <Windows.h>
#include <stdlib.h>
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "findHSValues.h"
#include "FindDirection.h"
#include <array>

using namespace std;
using namespace cv;

bool getImage(Mat& img)
{
	cout << "Enter image path: ";
	string path;
	cin >> path;
	img = imread(path, IMREAD_UNCHANGED);
	if (img.empty())
	{
		cout << "This image cant be loaded" << endl;
		return false;
	}
	return true;
}

int main()
{
	Mat origImg;
	while (!getImage(origImg))
	{
		continue;
	}

	cout << "Filter Irises" << endl;
	array<int, 8> eyeIrises = findHSValues(origImg);
	cout << "Filter Whites" << endl;
	array<int, 8> eyeWhites = findHSValues(origImg);


	while (!getImage(origImg))
	{
		continue;
	}
	findDirection(eyeWhites, eyeIrises, origImg);
	return 0;

}
```

Приложение 2 Фильтрующая функция

```
#include <iostream>
#include <string>
#include <Windows.h>
#include <stdlib.h>
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <array>

using namespace std;
using namespace cv;

static int done = 0;

static enum HSVNames
{
	HMIN,
	SMIN,
	VMIN,
	HMAX,
	SMAX,
	VMAX,
	ERODECOUNT,
	DILATECOUNT,
	MAXVAL
};

static void createTrackbars(array<int, 8> &val) {
	done = 0;
	const string trackbarWindowName = "Trackbars";
	namedWindow(trackbarWindowName, WINDOW_AUTOSIZE);
	createTrackbar("H_MIN", trackbarWindowName, &val[HMIN], val[HMAX]);
	createTrackbar("H_MAX", trackbarWindowName, &val[HMAX], val[HMAX]);
	createTrackbar("S_MIN", trackbarWindowName, &val[SMIN], val[SMAX]);
	createTrackbar("S_MAX", trackbarWindowName, &val[SMAX], val[SMAX]);
	createTrackbar("V_MIN", trackbarWindowName, &val[VMIN], val[VMAX]);
	createTrackbar("V_MAX", trackbarWindowName, &val[VMAX], val[VMAX]);

	createTrackbar("Erode", trackbarWindowName, &val[ERODECOUNT], 10);
	createTrackbar("Dilate", trackbarWindowName, &val[DILATECOUNT], 10);

	createTrackbar("Done", trackbarWindowName, &done, 1);
}

void printBiggerContourPixels(vector<vector<Point>> contours)
{
	bool isClosed = true;
	double buf = arcLength(contours[0], isClosed);
	int returnValue = 0;

	for (int i = 0; i < contours.size(); ++i)
	{
		double iteratedLength = arcLength(contours[i], isClosed);
		if (iteratedLength > buf)
		{
			buf = iteratedLength;
			returnValue = i;
		}
	}
	cout << "Iris found pixel count: " << contourArea(contours[returnValue]) << endl;
}
```

```
array<int, 8> findHSValues(Mat OrigImg)
{
	Mat img;
	OrigImg.copyTo(img);
	array<int, 8> HSVal{0, 0, 0, 256, 256, 256, 0, 0};
	createTrackbars(HSVal);
	Mat HSVimage, filtImage;
	cvtColor(img, HSVimage, COLOR_BGR2HSV);
	Mat erodeElement = getStructuringElement(MORPH_RECT, Size(2, 2));
	Mat dilateElement = getStructuringElement(MORPH_RECT, Size(2, 2));

	vector< vector<Point> > contours; //------------
	vector<Vec4i> hierarchy; //-------------
	while (!done)
	{
		OrigImg.copyTo(img);
		inRange(HSVimage, Scalar(HSVal[HMIN], HSVal[SMIN], HSVal[VMIN]),
Scalar(HSVal[HMAX], HSVal[SMAX], HSVal[VMAX]), filtImage);

		int buf = HSVal[ERODECOUNT];
		while (buf)
		{
			erode(filtImage, filtImage, erodeElement);
			--buf;
		}

		buf = HSVal[DILATECOUNT];
		while (buf)
		{
			dilate(filtImage, filtImage, dilateElement);
			--buf;
		}

		findContours(filtImage, contours, hierarchy, RETR_CCOMP,
CHAIN_APPROX_SIMPLE); //------------------
		drawContours(img, contours, -1, Scalar(0, 0, 255)); //----------------------
----

		namedWindow("Original Image", WINDOW_FREERATIO);
		namedWindow("Filtered Image", WINDOW_FREERATIO);
		imshow("Original Image", img);
		imshow("Filtered Image", filtImage);

		waitKey(30);
	}

	destroyAllWindows();

	printBiggerContourPixels(contours);

	return HSVal;

}
```

Приложение 3. Функция анализатор

```
#include <iostream>
#include <string>
#include <Windows.h>
#include <stdlib.h>
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <array>
#include <vector>
#include <cmath>

using namespace std;
using namespace cv;

static enum HSVNames
{
	HMIN,
	SMIN,
	VMIN,
	HMAX,
	SMAX,
	VMAX,
	ERODECOUNT,
	DILATECOUNT,
	MAXVAL
};

static enum ImportantPoints
{
	LEFTPT,
	RIGHTPT,
	UPPT,
	DOWNPT,
	MAXVALPT
};

Mat applyChanges(Mat img, array<int, 8> values)
{
	Mat ImgToReturn;
	inRange(img, Scalar(values[HMIN], values[SMIN], values[VMIN]),
Scalar(values[HMAX], values[SMAX], values[VMAX]), ImgToReturn);
	Mat erodeElement = getStructuringElement(MORPH_RECT, Size(2, 2));
	Mat dilateElement = getStructuringElement(MORPH_RECT, Size(2, 2));

	for (int i = 0; i < values[ERODECOUNT]; ++i)
		erode(ImgToReturn, ImgToReturn, erodeElement);

	for (int i = 0; i < values[DILATECOUNT]; ++i)
		dilate(ImgToReturn, ImgToReturn, dilateElement);

	return ImgToReturn;
}

int findBiggerContour(vector<vector<Point>> contours)
{
	bool isClosed = true;
	double buf = arcLength(contours[0], isClosed);
	int returnValue = 0;

	for (int i = 0; i < contours.size(); ++i)
	{
		double iteratedLength = arcLength(contours[i], isClosed);
		if (iteratedLength > buf)
```

```
			{
				buf = iteratedLength;
				returnValue = i;
			}
		}
		cout << "Iris found pixel count: " << contourArea(contours[returnValue]) << endl;
		return returnValue;
}

array<Point, 4> findImportantPoints(Mat img)
{
		vector< vector<Point> > contours;
		vector<Vec4i> hierarchy;
		array<Point, 4> returnValue;
		findContours(img, contours, hierarchy, RETR_CCOMP, CHAIN_APPROX_SIMPLE);
		int IndexContour = findBiggerContour(contours);

		int iXmin = 0;
		int iXmax = 0;
		int iYmin = 0;
		int iYmax = 0;

		for (int i = 0; i < contours[IndexContour].size(); ++i)
		{
			if (contours[IndexContour][i].x < contours[IndexContour][iXmin].x)
				iXmin = i;

			if (contours[IndexContour][i].x > contours[IndexContour][iXmax].x)
				iXmax = i;

			if (contours[IndexContour][i].y < contours[IndexContour][iYmin].y)
				iYmin = i;

			if (contours[IndexContour][i].y > contours[IndexContour][iYmax].y)
				iYmax = i;
		}

		returnValue[LEFTPT] = contours[IndexContour][iXmin];
		returnValue[RIGHTPT] = contours[IndexContour][iXmax];
		returnValue[UPPT] = contours[IndexContour][iYmin];
		returnValue[DOWNPT] = contours[IndexContour][iYmax];

		return returnValue;
}

void printDirections(array<bool, 4> checker)
{
		cout << "Eye's direction is ";

		if (checker[LEFTPT] && checker[RIGHTPT])
			cout << "Center ";
		else if (checker[LEFTPT] && !checker[RIGHTPT])
			cout << "Left (Right on a picture) ";
		else if (!checker[LEFTPT] && checker[RIGHTPT])
			cout << "Right (Left on a picture) ";
		else if (!checker[LEFTPT] && !checker[RIGHTPT])
			cout << "Somewhere (probably an error) ";
		else
		{
			cout << "Something went wrong";
			return;
		}

		if (checker[UPPT] && checker[DOWNPT])
```

```
			cout << "forward" << endl;
		else if (checker[UPPT] && !checker[DOWNPT])
			cout << "down" << endl;
		else if (!checker[UPPT] && checker[DOWNPT])
			cout << "up" << endl;
		else if (!checker[UPPT] && !checker[DOWNPT])
			cout << "forward (eye is closed slightly)" << endl;
		else
		{
			cout << "Something went wrong";
			return;
		}
}

void findAndPrintDirection(Mat whitesImg, array<Point, 4> importantPoints)
{
		int radius = fabs(importantPoints[LEFTPT].x - importantPoints[RIGHTPT].x)/2;
		int minReqRange = fabs(importantPoints[LEFTPT].x - importantPoints[RIGHTPT].x) *
0.2;
		int buf = 0;
		array<bool, 4> checker{ 0, 0, 0, 0 };
		cvtColor(whitesImg, whitesImg, COLOR_GRAY2BGR);

		for (int i = importantPoints[LEFTPT].x; i > importantPoints[LEFTPT].x - radius; --
i)
		{
			if (whitesImg.at<Vec3b>(i, importantPoints[LEFTPT].y) == Vec3b(255, 255,
255))
			{
				++buf;
			}

			if (buf == minReqRange)
			{
				checker[LEFTPT] = 1;
				break;
			}
		}

		buf = 0;
		for (int i = importantPoints[RIGHTPT].x; i < importantPoints[RIGHTPT].x + radius;
++i)
		{
			if (whitesImg.at<Vec3b>(i, importantPoints[RIGHTPT].y) == Vec3b(255, 255,
255))
			{
				++buf;
			}

			if (buf == minReqRange)
			{
				checker[RIGHTPT] = 1;
				break;
			}
		}


		minReqRange = fabs(importantPoints[LEFTPT].x - importantPoints[RIGHTPT].x) * 0.12;
		buf = 0;
		for (int i = importantPoints[UPPT].y; i < importantPoints[UPPT].y - radius; --i)
		{
			if (whitesImg.at<Vec3b>(importantPoints[UPPT].x, i) == Vec3b(255, 255,
255))
			{
```

```
                ++buf;
            }

            if (buf == minReqRange)
            {
                checker[UPPT] = 1;
                break;
            }
        }


        buf = 0;
        for (int i = importantPoints[DOWNPT].y; i < importantPoints[DOWNPT].y + radius;
++i)
        {
            if (whitesImg.at<Vec3b>(importantPoints[DOWNPT].x, i) == Vec3b(255, 255,
255))
            {
                ++buf;
            }

            if (buf == minReqRange)
            {
                checker[DOWNPT] = 1;
                break;
            }
        }

        printDirections(checker);

}

void visualiseProcess(Mat& img, array<Point, 4> importantPoints)
{
        circle(img, importantPoints[LEFTPT], 1, Scalar(0, 0, 255), 1);
        circle(img, importantPoints[RIGHTPT], 1, Scalar(0, 255, 0), 1);
        circle(img, importantPoints[UPPT], 1, Scalar(0, 255, 255), 1);
        circle(img, importantPoints[DOWNPT], 1, Scalar(255, 0, 0), 1);

        int minReqRange = fabs(importantPoints[LEFTPT].x - importantPoints[RIGHTPT].x) *
0.2;
        line(img, importantPoints[LEFTPT], Point(importantPoints[LEFTPT].x - minReqRange,
importantPoints[LEFTPT].y), Scalar(255, 0, 255));
        line(img, importantPoints[RIGHTPT], Point(importantPoints[RIGHTPT].x +
minReqRange, importantPoints[RIGHTPT].y), Scalar(255, 0, 255));

        minReqRange = fabs(importantPoints[LEFTPT].x - importantPoints[RIGHTPT].x) * 0.12;
        line(img, importantPoints[UPPT], Point(importantPoints[UPPT].x,
importantPoints[UPPT].y - minReqRange), Scalar(255, 255, 0));
        line(img, importantPoints[DOWNPT], Point(importantPoints[DOWNPT].x,
importantPoints[DOWNPT].y + minReqRange), Scalar(255, 255, 0));
}

void findDirection(array<int, 8> whites, array<int, 8> irises, Mat img)
{
        Mat HSVimage;
        cvtColor(img, HSVimage, COLOR_BGR2HSV);

        Mat whitesImg = applyChanges(HSVimage, whites);
        Mat irisesImg = applyChanges(HSVimage, irises);
        array<Point, 4> importantPoints = findImportantPoints(irisesImg);

        visualiseProcess(img, importantPoints);
```

```
	vector<vector<Point>> contours;
	findContours(whitesImg, contours, RETR_CCOMP, CHAIN_APPROX_SIMPLE);
	drawContours(img, contours, -1, Scalar(255, 0, 0));

	findContours(irisesImg, contours, RETR_CCOMP, CHAIN_APPROX_SIMPLE);
	drawContours(img, contours, -1, Scalar(0, 255, 0));

	namedWindow("whites", WINDOW_FREERATIO);
	namedWindow("irises", WINDOW_FREERATIO);
	namedWindow("Orig", WINDOW_FREERATIO);

	imshow("whites", whitesImg);
	imshow("irises", irisesImg);
	imshow("Orig", img);

	waitKey(30);

	//findAndPrintDirection(whitesImg, importantPoints);



	while (1)
	{
		imshow("whites", whitesImg);
		imshow("irises", irisesImg);
		imshow("Orig", img);

		waitKey(30);
	}

}
```