МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Ярославский государственный университет им. П.Г. Демидова»

Кафедра компьютерной безопасности и математических методов обработки информации

Курсовая работа

Разработка системы альтернативного управления компьютером.

(Специальность 10.05.01 Компьютерная безопасность)

Научный руководитель

Д-р наук, профессор
(степень, звание)
В.Г. Дурнев
(подпись) (ФИО)
«___» _____ 20__ г.

Студент группы КБ-51СО
С.М. Соловьев
(подпись) (ФИО)
«___» _____ 20__ г.

Ярославль 2017 г.

Оглавление

Введение

Человек на протяжении своей жизни, по разным данным, получает от 80 % до более 90% информации с помощью зрения. Все привыкли считать глаза неким «устройством» для получения информации, но мало кто задумывался о том, что глаза также могут передавать информацию.

Несколько лет назад такое предложение могло показаться странным, но теперь технологии шагнули далеко вперед, и это стало реальностью. С помощью глаз можно передавать информацию о том, куда смотрит человек, на чем сфокусирован его взгляд, на чем он сосредоточен.

Использовать данную технологию можно для обучения маленьких детей или детей с заторможенным развитием. На экране одновременно отображаются несколько изображений. Обучаемого просят посмотреть на какое-то конкретное по разным характеристикам (цвет, вид, форма и т.д.). Также маркетологи могут измерять эффективность рекламы, демонстрируя ее людям и фиксировать координаты объекта внимания человека. Но, пожалуй, самая главное направление использования подобной технологии – это помощь людям с ограниченными физическими возможностями в использовании ПК для общения, саморазвития, работы или управления другими устройствами.

Данный проект создан для реализации приложения с открытым исходным кодом и бесплатной лицензией на использование и распространение продукта.

Цель работы

1. Спроектировать приложение для управления ПК взглядом.
2. Разработать прототип приложения.
3. Спланировать дальнейшее развитие приложения.

Введение в биологическую терминологию

Используя ПК или мобильное устройство, человек устремляет взгляд на конкретные секторы экрана. Фиксацию взгляда на конкретном секторе будем называть фокусом взгляда. Для управления ПК требуется постоянный трекинг (слежение) фокуса взгляда. Для понимания физики процесса требуется информация о биологии и поведении глаза. Данный раздел предоставит некоторые базовые знания для создания системы глазного трекинга.

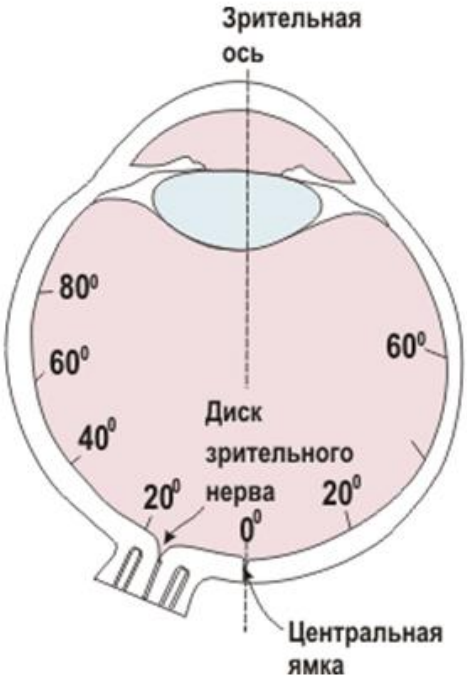
Структура человеческого глаза

Для того чтобы человек мог максимально четко рассмотреть объект, ему требуется сфокусировать на нем взгляд. Получение изображения происходит при попадании его через хрусталик на рецепторы глаза. Важным фактором при рассмотрении структуры глаза является то, где располагаются эти рецепторы. В [2] говорится, что если бы рецепторы были расположены равномерно внутри глаза, то была бы проблема понять, как происходит фокусирование. Вместо этого мы имеем различную плотность распределения рецепторов. Наиболее четкое изображение человек получает при попадании его в точку максимального скопления фоторецепторов-колбочек, и эта точка находится в желтой ямке (пятно фовеа).

Фиксация глаза

Но, нельзя забывать, что человеческий зрачок не может замереть на месте для получения стабильного изображения. В [1] используют понятие дрейфа. Дрейф представляет собой неупорядоченное и относительно медленное движение осей глаз, при котором для каждого глаза изображение точки фиксации остается внутри фовеа. Дрейф всегда сопровождается тремором – высоким по частоте, но очень маленьким по амплитуде колебательным движением осей глаз. Проще говоря, глаз даже при фокусировке изображения немного дергается, незаметно для самого человека.

Так как желтая ямка находится внутри глаза и имеет крайне малый размер (диаметр чуть больше 1мм), то фиксировать изменение ее положения не предоставляется возможным. Вместо этого можно отслеживать зрачок и через него выводить точку фокусировки, это становится возможным потому что, объект фокуса находится в центре глаза.



Локализация центральной ямки (0 градусов) и различных областей сетчатки по отношению к ней (в градусах эксцентриситета).

Перемещения глаза

Человеческий глаз может прибывать только в двух состояниях: фокусировка, сопровождающаяся дрейфом и перемещение (саккада).

Саккады (от старинного французского слова, переводимого как «хлопок паруса») — быстрые, строго согласованные движения глаз, происходящие одновременно и в одном направлении. В работе [3] было обнаружено, что «трансаккадная пространственная память приносится в жертву, чтобы поддерживать перцептивную стабильность»[3]. Это означает, что во время саккадного движения визуальная информация не получается,

чтобы сохранить восприятие гладкости и стабильности. Этот эффект известен как саккадическое подавление. Подобным эффектом обладает постоянное использование мыши в качестве интерфейса управления ПК, оно оттачивает навык до сглаженных, точных перемещений курсора в нужное положение.

Саккады важно отличать от других движений глаз, называемых Saccadic Eye Movements (SACs) в [5] (Преследование, вергентное и вестибулярное). Преследование – движения глаз «преследуют» точку фокусировки, перемещающийся в пространстве, выполняя саккады, чтобы догнать, если точка движется слишком быстро. Движение заметно медленнее, чем у саккад. Вергентные движения (лат. *Vergo* - склоняюсь) - макродвижения глаз, приводящие к изменению угла между зрительными осями левого и правого глаза. Вергентные движения – выступают фактором, обеспечивающим бинокулярное зрение. Они разделяются на движения конвергенции – сводящие глаза, дивергенции –разводящие глаза, циклофузионные, или торсионные движения, меняющие ориентации сетчаток левого и правого глаз. Вестибулярные движения - это вращения глаза, чтобы компенсировать большие движения головы или тела (стабилизация изображения). Эти три движения менее важны, чем саккады для обработки информации, и как таковые для отслеживания зрачков.

Методы поиска глаз

Самые точные методы для обнаружения глаз включают в себя инвазивное или дорогостоящие оборудование. В [2] упоминается метод, который использует контактную линзу, с металлическим кольцом. Фиксация координат происходит с помощью измерения магнитного поля. В [6] в качестве системы фиксации изображения используют шлем с ИК камерами. Минусы данных систем заключаются в том, что они неудобны для использования, не мобильны.

Трудности, возникающие в процессе разработки подобных систем неизбежны. В настоящее время исследования в этой области направлены на создание чувствительного к пользователю и легкодоступного устройства. Идеальным решением для этого является бесконтактное и неинвазивное устройство. Но с такими устройствами возникают новые трудности, такие как алгоритмы поиска головы, глаз, зрачка. Решения данных проблем будут рассмотрены далее.

Инфракрасное отражение

Принцип работы метода ИК отражения заключается в том, что повышается контраст между зрачком и радужной оболочкой. Ниже приведен пример использования инфракрасного отражения для обнаружения глаз.

В [7] авторы сначала определяют область 60x60 пикселей (исходное разрешение монитора 1280 × 1024 пикселей) с ориентировочной оценкой центра зрачка. Поскольку глаза имеют разное расположение от источника ИК света, они имеют разные показатели преломления, зрачок кажется черным, а остальное нет. Использование простого порогового алгоритма для обработки, отфильтрованной серой шкалы изображения, полученного с камеры, проверяются все точки ниже для определения ROI. Пороговое уравнение, использованное в [7], приведено ниже.

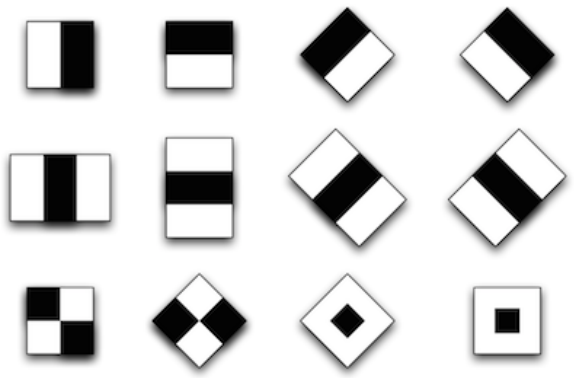
$$X_{pupil} = \frac{1}{N}\sum_{n=1}^{N} x_n$$
$$Y_{pupil} = \frac{1}{N}\sum_{n=1}^{N} y_n$$

Общая идея использования инфракрасного излучения заключается в том, что отражение инфракрасного излучения упрощает процесс определения центра зрачка. Таким образом, уменьшается сложность расчета и повышается точность обнаружения зрачков, повышается эффективность системы отслеживания глаз. Отрицательная сторона этого заключается в потребности инфракрасного света. Также могут быть другие источники света, или тени, которые нарушают это изображение и создают «шум», что усложняет вычисление, но это проблема актуальна для большинства ненавязчивых систем.

Каскад Хаара

Признаки Хаара — признаки цифрового изображения, используемые в распознавании образов. Своим названием они обязаны интуитивным сходством с вейвлетами Хаара. Признаки Хаара использовались в первом детекторе лиц, работающем в реальном времени.

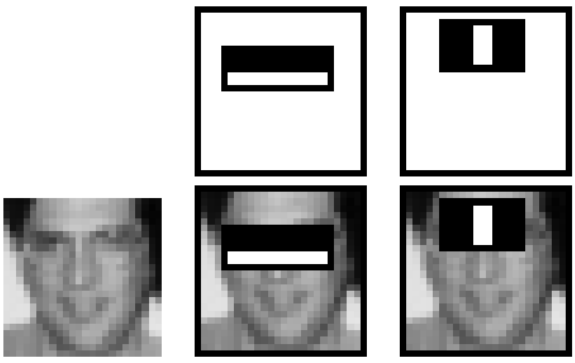
Обычно у каждого метода есть основа, то, без чего этот метод не мог бы существовать в принципе, а уже над этой основой строится вся остальная часть. В методе Виолы-Джонса эту основу составляют примитивы Хаара, представляющие собой разбивку заданной прямоугольной области на наборы разнотипных прямоугольных подобластей:



В оригинальной версии алгоритма Виолы-Джонса использовались только примитивы без поворотов, а для вычисления значения признака сумма яркостей пикселей одной подобласти вычиталась из суммы яркостей другой подобласти [8]. В развитии метода были предложены примитивы с наклоном на 45 градусов и несимметричных конфигураций. Также вместо вычисления обычной разности, было предложено приписывать каждой подобласти определенный вес и значения признака вычислять как взвешенную сумму пикселей разнотипных областей [9]:

$$feature = \sum_{i \in I = 1 \dots N} w_i * RectSum(r_i)$$

Основной причиной, почему в основу метода легли примитивы Хаара, являлась попытка уйти от пиксельного представления с сохранением скорости вычисления признака. Из значений пары пикселей сложно вынести какую-либо осмысленную информацию для классификации, в то время как из двух признаков Хаара строится, например, первый каскад системы по распознаванию лиц, который имеет вполне осмысленную интерпретацию [8]:



Сложность вычисления признака, так же как и получения значения пикселя, остается O(1): значение каждой подобласти можно вычислить скомбинировав 4 значения интегрального представления (Summed Area Table — SAT), которое в свою очередь можно построить заранее один раз для всего изображения за O(n), где n — число пикселей в изображении, используя формулу [9]:

$$SAT(x,y) = SAT(x,y-1) + SAT(x-1,y) + I(x,y) - SAT(x-1,y-1)$$

$$SAT(-1,y) = SAT(x,-1) = SAT(-1,-1) = 0$$

Таким образом, создали быстрый алгоритм поиска объектов, который пользуется успехом уже больше десятилетия. Но вернемся к нашим признакам. Для определения принадлежности к классу в каждом каскаде, находится сумма значений слабых классификаторов этого каскада. Каждый слабый классификатор выдает два значения в зависимости от того больше или меньше заданного порога значение признака, принадлежащего этому классификатору. В конце сумма значений слабых классификаторов сравнивается с порогом каскада, и выносится решение, найден объект данным каскадом или нет.

Для разных видов объектов создаются разные XML классификаторы. Существуют базы и приложения, с помощью которых можно получить XML классификатор объекта. Мы воспользуемся каскадом, полученным студентами из Modesto Castrillon-Santana (IUSIANI, University of Las Palmas de Gran Canaria, Spain) в 2006 году. Лицензия открыта для некоммерческого использования.

Пример структуры описания:

```
<_>
  <!-- tree 0 -->
  <_>
    <!-- root node -->
    <feature>
      <rects>
        <_>
          7 3 30 3 -1.</_>
        <_>
          17 3 10 3 3.</_></rects>
      <tilted>0</tilted></feature>
    <threshold>0.1012997999787331</threshold>
    <left_val>-0.7954636812210083</left_val>
    <right_val>0.7811083793640137</right_val>
  </_>
</_>
```

Создается впечатление, что здесь куча непонятных цифр и странной информации, но на самом деле все просто:

- Дерево объектов — набор слабых классификаторов, на основе которых выносится решение о том, находится объект на изображении или нет,
- Два объекта в узле (без имени) — это параметры конкретного слабого классификатора.
    - Расшифровка первого объекта слева направо: первые два значения в нашем случае не используются, третье — номер признака в общей таблице признаков (она располагается дальше в XML), четвертое — пороговое значение слабого классификатора. Так как у нас используется классификатор, основанный на одноуровневых решающих деревьях (Decision Stump), то если значение признака Хаара меньше порога слабого классификатора (четвертое значение), выбирается значение left_val, если больше — right_val.

Все эти признаки в какой-то степени являются самыми обыкновенными детекторами границ. На основе этого базиса вычисляется решение о том, распознал ли каскад объект на изображении или нет. Второй по важности момент в методе Виола-Джонса — это использование каскадной модели или вырожденного дерева принятия решений: в каждом узле дерева с помощью каскада принимается решение, содержится объект на изображении или нет. Если объект не содержится, то алгоритм заканчивает свою работу, если он может содержаться, то мы переходим к следующему узлу. Обучение построено таким образом, чтобы на начальных уровнях с наименьшими затратами отбрасывать большую часть окон, в которых не может содержаться объект. В случае распознавания лиц — первый уровень содержит всего 2 слабых классификатора, в случае распознавания глаз — 6.

Поскольку каскады Хаара не требуют специального оборудования и технически реализуются достаточно быстро, выбор пал на эту технологию поиска глаз.

Вывод

Чтобы отслеживать взгляд пользователя, нам нужно знать, где находится зрачок. Системе не требуется чрезмерно мощная камера, поскольку фиксация глаза – единственные значимые данные, которые требуются, чтобы найти взгляд, и они соответствуют самому длинному периоду времени отсутствия движения глаза. Существует большое количество доступных для использования алгоритмов, в которых используется большее различных методов. Основными требованиями являются:

- обнаружение глаз,
- обнаружение зрачка,
- обнаружение дополнительной функции,
- преобразование в координаты фокуса.

Алгоритмы распознавания взгляда

Система отслеживания взглядов состоит из сложной комбинации алгоритмов для получения точных координат взгляда. Поиск глаз на изображении – это только часть. Существует много способов комбинирования методов для получения требуемого результата. Рассмотрим некоторые примеры алгоритмов отслеживания взгляда.

Нейронные сети

Первым шагом использования нейронных сетей, согласно [11], является обнаружение лица и глаз. Глаз и лицо обнаруживаются с использованием интенсивностей пикселей в области, а затем каждому региону дается оценка. Далее оценки упорядочиваются иерархически и подаются в OpenCV библиотеку для обнаружения. Следующий шаг – уменьшить ROI на уменьшенную картинку, обычно она чрезвычайно малого размера, меньше одной тысячной от размера исходного изображения, и содержит только глаз.

Данное изображение обрабатывается, чтобы сделать более ясный вход для нейронной сети. Применяются два процесса:

1. выравнивание гистограммы, которое осветляет склеру и затемняет границы глаз,
2. изменение размера изображения до меньшего формата с использованием бикубической интерполяции. Это уменьшает пиксель в четыре раза, что сокращает время обучения Нейронной сети от нескольких минут до, как правило, менее 60 секунд. [11]

Интенсивность каждого пикселя изображения используется как вход для нейронной сети. Нейронная сеть использует прямую подачу и двухслойную структуру. Работа связана с концепциями искусственного интеллекта, а не с обработкой изображений.

Ошибка, найденная с использованием подхода Нейронных сетей в [11], оказалась примерно 1,5 градуса.

Нейронная сеть используется по-разному, например в [12], первоначально система использовала обнаружение мигания для точного определения положения глаз. Как только глаза обнаружены, обнаружение радужной оболочки использует комбинацию обнаружения кромок и округлых преобразований Хафа, за которым следует угловое обнаружение, направленное на обнаружение как внутреннего, так и внешнего углов. В данном примере Нейронная сеть используется только для сопоставления связи между чертами лица и направлением взгляда. Отличается такой подход от используемого в [11], тем, что обнаружение функции выполняется без Нейронной сети.

Оба алгоритма для Нейронной сети были эффективны, но нейронная сеть добавила сложности к системам, теперь их нужно обучить, что увеличивает время калибровки.

Алгоритм отслеживания взгляда в реальном времени с компенсацией движения головы(RTAC)

Объяснение RTAC в [10] начинается со ссылкой на «Pupil Center Corneal Reflection (PCCR) »[10, стр. 395]. Согласно PCCR, вычисления направления взгляда сначала приобретают вектор блика зрачка, а затем используется функция отображения взгляда. Блик является отражением инфракрасного источника света в глазу. Вектор блика зрачка является двумерным вектором между бликом и зрачком. Затем алгоритм использует функцию для отображения этого вектора к направлению 3D-взгляда. Функция отображения определяется следующими функциями:

$$\theta_h = b * \theta_h * V_x + a * \theta_h$$

$$\theta_v = b * \theta_v * V_y + a * \theta_v$$

Где θ_h – угол между направлением взгляда и горизонтальным направлением, а также θ_v – угол между направлением взгляда и вертикальным направлением. Коэффициенты, a и b, оцениваются с использованием наборов пар векторов бликов. Следующим шагом в этом алгоритме является вычисление движения головы. Предыдущий расчет не включает адаптацию к движениям головы. Таким образом, система должна компенсировать эти движения головы. Этот расчет довольно сложен. Основа для этого два уравнения:

$$\Delta I_x = (b_{2x}\Delta L_v + a_{2x})\Delta d^2 + (b_{1x}\Delta L_h + a_{1x})\Delta d + (b_{0x}\Delta L_h + a_{0x})$$

$$\Delta I_y = (b_{2y}\Delta L_v + a_{2y})\Delta d^2 + (b_{1y}\Delta L_v + a_{1y})\Delta d + (b_{0y}\Delta L_v + a_{0y})$$

Эти два уравнения описывают изменения в положении головы в горизонтальном и вертикальном положении соответственно. Вывод этих уравнений можно найти в [10, стр. 397].

Метод компенсации, используемый в этом алгоритме, использует два набора уравнений:

1. Вычисление пропорционального изменения увеличения и последующее использование этих значений для вычисления значения компенсации в горизонтальном и вертикальном направлениях. [10, стр. 397].
2. Затем следующим шагом является фиксация головы пользователя во время калибровки, чтобы получить начальные требуемые параметры. После этой калибровки пользователь может свободно перемещать свою голову и параметры сравниваются, чтобы выполнить вычисления.

В [10] было обнаружено, что этот алгоритм имел точность порядка 1 градуса, И что между разными предметами мало что изменилось. Существовала также небольшая разница между различными положениями

головы, но ошибка действительно возрастала, когда голова перемещалась около границ видения камеры.

Концепция приложения

Для реализации проекта потребуется веб-камера и открытая библиотека Computer Vision OpenCV. Поскольку логическая конструкция большинства систем трекинга является довольно стандартной, общий дизайн этой системы схож с существующими системами. Однако инструменты, используемые в этих системах, имеют большое количество вариаций.

Технические проблемы

Традиционные бесконтактные системы отслеживания глаз имеют ряд проблем, которые необходимо учесть при расчете фокуса или при нахождении глаза.

Первая и самая главная проблема, это движение головы, о котором упоминалось в [11]. Старые системы отслеживания глаз требуют, чтобы пользователь продолжал удерживать голову неподвижно, чтобы точно зафиксировать данные. Алгоритмы, рассмотренные ранее и включающие в себя инфракрасное освещение, не поддерживают изменение положения головы в кадре.

Вторая проблема, с которой сталкивается область глазного трекинга, известна как проблема «Midas Touch». Эта проблема является прямым следствием движения глаз. Глаза используются как устройства ввода информации и поэтому нужно точно определять перемещение ли это или мелкие колебания (дрейфы). Так же пользователь должен уметь отправлять команды, данная проблема решена в предыдущей работе по созданию альтернативной системы управления ПК[13]. Для решения данной проблемы используется постоянно доступный контроль вверху экрана с набором команд (клик, двойной клик, правый клик, скролл). Для выбора команды нужно задержать курсор на некоторое время (имеется возможность индивидуальной настройки, в среднем 1сек). Далее курсор наводится на нужный элемент экрана и задерживается на то же время, чтобы совершилось

действие. Так же учитывается наличие дрейфов и трудности задержать взгляд на одном месте и настраивается радиус, в котором таймер не прекращает отсчет времени. Другой путь решения данной проблемы является явный вынос команд на клавиши (реализуемо для малоподвижных пользователей) или моргания глазом.

Язык программирования

Поскольку для выполнения задачи требуется библиотека OpenCV, то ее совместимость ограничивает выбор между такими языками как Python, Java и .Net семейство. Выбор пал на C#, так как данный язык более знаком мне, и имплементация взаимодействия с курсором ОС частично реализована в предыдущей работе. В дальнейшем рассматривается вариант решения на Python для кроссплатформености.

Веб-камера

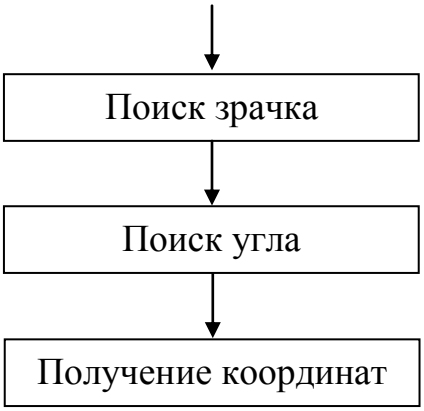
В качестве веб-камеры можно использовать встроенную камеру в ноутбук или любую другую подключаемую к ПК камеру.

Алгоритм

Общий алгоритм построен на основе непрерывной обработке потока изображения.

Шаги цикла – это шаги, необходимые для получения всех необходимых данных из одного кадра.

Простейшая форма этой конструкции показана ниже:



Поиск пары глаз объясняется тем, что это более быстрая операция, нежели поиск каждого глаза по отдельности. Полученное изображение пары глаз делится ровно пополам для обработки каждого глаза отдельно. Оба глаза используются для повышения точности системы. Как только глаза получены, необходимы зрачок и углы. Методом обнаружения зрачка является обнаружение кругов на специально обработанном изображении, далее будет подробно раскрыт алгоритм обработки. Также необходимо уменьшить шум в изображении. Пороговая методика полезна для уменьшения шума изображения. Как только изображение максимально очистится от шумов, тогда можно искать круги и найти зрачок. Центр зрачка можно аппроксимировать как центр этого обнаруженного круга. Внутренний угол глаза был выбран как вторая функция для проверки. Причина этого связана с тем, что он успешно использовался в [12], где создавался вектор для отслеживания изменений в глазу, когда он перемещается внутри глазницы. Если координаты глаза и угла получены для обоих глаз, то они передаются в компонент управления курсором. Шаг цикла завершен.

Калибровка

Целью алгоритма калибровки является поиск контрольных точек, которые впоследствии могут быть использованы для перевода положения зрачка и угловых позиций в положение на экране. Техника калибровки,

используемая в подобных приложениях, представляет собой черный экран с одиночной зеленой точкой. Пользователь фиксирует взгляд на точке, чтобы записать координаты. Затем появляется следующая точка, так продолжается, пока не будет записано двенадцать точек. Эта система получения эталонных координат полезна в простой системе, подобной разрабатываемой, потому что она обеспечивает опорные координаты внешней части экрана, а также средних опорных точек. Таким образом, система использует систему точечной калибровки, чтобы получить опорные координаты. Тем не менее, используются только девять точек, так как это разбивает экран на четыре области. Если требуются дополнительные области, в калибровке требуется больше точек. Координаты каждой точки также записываются несколько раз, чтобы помочь уменьшить вероятность ошибки.

Управление ПК

После завершения калибровки начинается процесс трекинга. Результаты сохраняются в основной системе калибровки, а затем начинается цикл шагов. Разница в шагах цикла заключается в том, что добавлено несколько дополнительных шагов. Вычисляется угол и координаты зрачков, далее используя простые сравнения расстояний между точками и текущий фокус, область фокуса находится. Как только область будет сведена к одному из четырех частей экрана, расстояния преобразуются в координаты в пределах экрана.

Для проверки точности системы основной контур включает в себя рисование одной точки в одной из четырех позиций, расположенных в каждой из частей экрана. Как только эта точка была посчитана, все полученные координаты после этого сравниваются с координатами точки для записи ошибки. Ошибка записывается в файл по мере ее обнаружения.

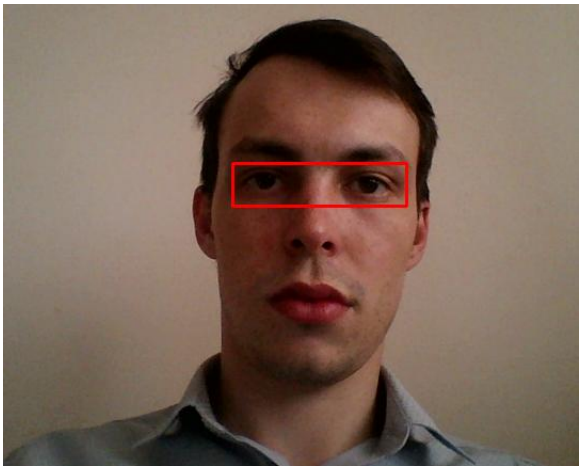
Реализация проекта

Поиск пары глаз

Как уже было сказано ранее, поиск пары глаз осуществляется с помощью каскадов Хаара. Библиотека OpenCV поддерживает работу с классификаторами Хаара, данный функционал встроен в библиотеку. В ходе разработки было выяснено, что лучше использовать классификаторы Хаара следующим образом:

1) Классификаторы нужно загрузить на этапе инициализации программы
2) Изображение требуется подготовить перед запуском поиска пары глаз. Изображение приводится к черно-белому виду, так как в данном случае классификаторы отрабатывают более устойчиво.

Метод обнаружения Хаара принимает на вход изображение, каскад классификаторов Хаара, переменную для хранения результата, а затем несколько других необязательных параметров. Затем эта функция возвращает последовательность обнаруженных граней, состоящая из координат x и y, а также ширина и высота найденной пары глаз. Для более наглядного тестирования, вокруг пары глаз рисуется прямоугольник. Для проверки работы алгоритма, будем считать, что находится всегда только одна пара глаз, именно та, которая и будет управлять системой.

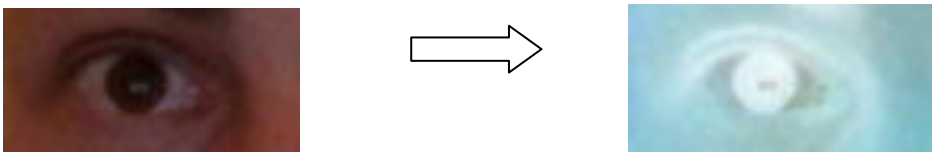


Пример обнаружения пары глаз

Определение зрачка

Изображение пары глаз делится ровно пополам. Далее будет разъяснена реализация получения координат для одного глаза, но эта реализация справедлива и для другого.

Изначально изображение подвергается инверсии (изображение изначально цветное). Инверсия производится с помощью библиотеки AForge уже знакомой из [13]. Результат инверсии на изображении ниже.



Исходное изображение Инвертированное изображение

Бинаризация изображения

Следующий шаг – это бинаризация изображения, использован для этого алгоритм Брэдли. Процесс бинаризации – это перевод цветного (или в градациях серого) изображения в двухцветное черно-белое. Главным параметром такого преобразования является порог t – значение, с которым сравнивается яркость каждого пикселя. По результатам сравнения, пикселю присваивается вес 0 или 1. Существуют различные методы бинаризации, которые можно условно разделить на две группы – глобальные и локальные. В первом случае величина порога остается неизменной в течение всего процесса бинаризации. Во втором изображение разбивается на области, в каждой из которых вычисляется локальный порог.

Существуют разные виды алгоритмов бинаризации изображения (метод Ниблэка, метод Кристиана). Наиболее быстрым из классических локальных алгоритмов считается метод Ниблэка, но он плохо работает с низкоконтрастными неровностями фона, которые в случае изображения глаза и близлежащего фона заполняют изображение чуть меньше, чем полностью. Чтобы исправить этот недостаток, было разработано несколько алгоритмов

на основе метода Ниблэка, называемых „ниблэковскими“. В качестве примера, метод Кристиана, показывал хорошие результаты, и рассматривался как подходящий вариант. Однако на изображении с нестабильным расстоянием от камеры, после применения этого алгоритма появлялись искажения.

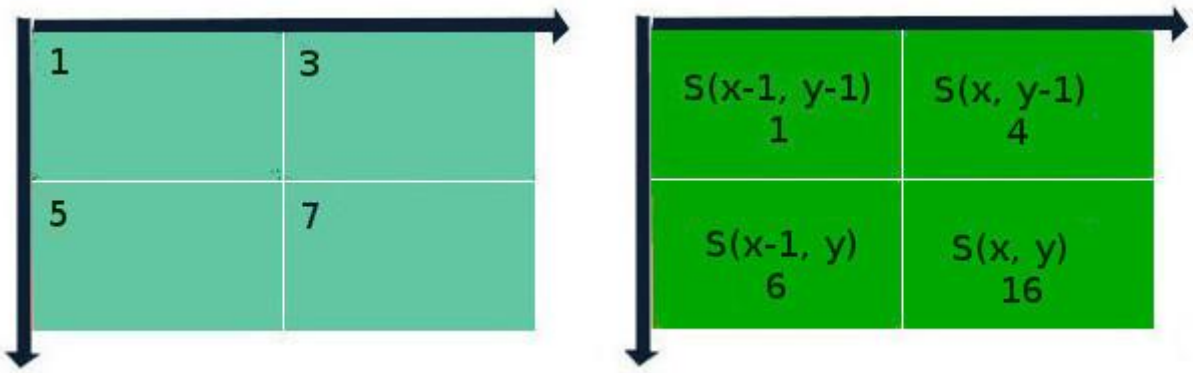
Трудность алгоритма бинаризации заключается в нахождении порогового значения, которое должно максимально надежно отделять символы не только от фона, но и от шума, теней, бликов и подобного. Часто для этого прибегают к методам математической статистики. В методе Брэдли подход с другой стороны – со стороны интегральных изображений.

Интегральные изображения – это не только эффективный и быстрый (всего за один проход по изображению) способ найти сумму значений пикселей, но и простой способ найти среднее значение яркости в пределах заданного участка изображения.

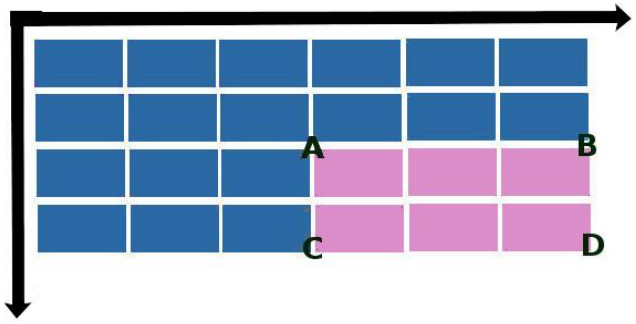
Допустим, у нас есть 8-битное изображение в оттенках серого (цветное изображение можно перевести в оттенки серого, пользуясь формулой $I = 0.2125R + 0.7154G + 0.0721B$). В таком случае, значение элемента интегрального изображения рассчитывается по формуле:

$$S(x, y) = I(x, y) + S(x - 1, y) + S(x, y - 1) - S(x - 1, y - 1);$$

где S – результат предыдущих итераций для данной позиции пикселя, I – яркость пикселя исходного изображения. Если координаты выходят за пределы изображения, они считаются нулевыми. Для понимания принцип работы представлен на схеме ниже:



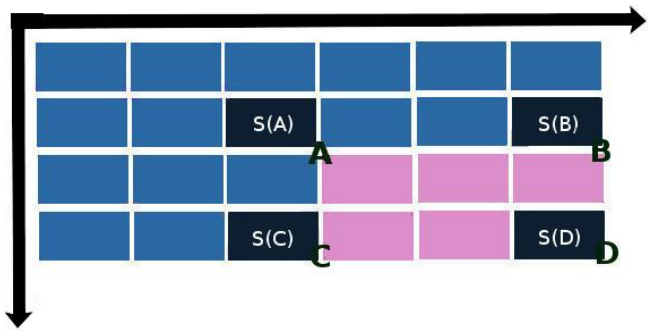
Хитрость алгоритма состоит в том, что, один раз составив интегральную матрицу изображения, можно быстро вычислять сумму значений пикселей любой прямоугольной области в пределах этого изображения. Пусть ABCD – интересующая нас прямоугольная область.



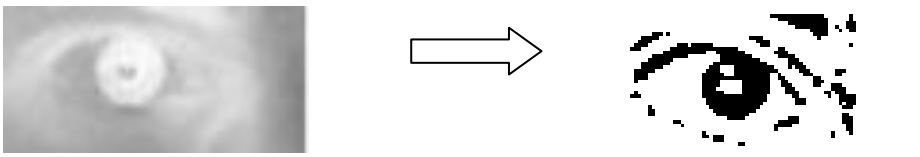
Тогда суммарная яркость S в этой области вычисляется по формуле:

$$S(x,y) = S(A) + S(D) - S(B) - S(C)$$

где S(A), S(B), S(C) и S(D) – значения элементов интегральной матрицы в направлении на „северо-запад“ от пересечений сторон прямоугольника:



Реализацию метода можно посмотреть в листинге программы. Результат бинаризации:



Инвертированное (ЧБ) изображение

Бинаризированное изображение

Применение бинаризации объясняется тем, что в таком случае на определения зрачка не влияет контраст и яркость изображения. Для того что б проверить данное утверждение был реализован модуль изменяющий контраст и яркость изображения. Результатом такой проверки стало подтверждение высказывания. Для дальнейшего снижения шума применяется сглаживающий фильтр Гаусса.

Следующий этап – поиск окружностей на изображении. Используемая для этого функция называется cvHoughCircles. Она ищет в изображении форму эллипса, которая упомянутая в нескольких работах. Чтобы функция обнаруживала только лучший круг на изображении, требуется правильно установить самый важный параметр – минимальное расстояние до размера изображения. Этот параметр представляет собой минимальное расстояние между двумя обнаруженными кругами. Для наглядности на изображении зрачка отображаются круги (те, что находит метод cvHoughCircles).

Поиск угла

Поскольку оба глаза обнаружены, и обнаружение углов использует ROI (Region Of Interest) от обнаружения глаз, то их достаточно просто вычислить. Угловое детектирование работает, уравнивая гистограмму изображения, а после вызывается метод OpenCV cvGoodFeaturesToTrack. Эта функция находит наиболее заметные углы изображения. Функция имеет параметр с именем maxCorners, который позволяет пользователю указать количество возвращаемых углов. Функция также оценивает углы по силе, поэтому,

выбираем maxCorners = 1, ведь нужно получить только самый сильный угол. Проблема была быстро обнаружена, самый сильный угол в полном изображении глаза может быть либо глазным углом, либо ошибочным углом, возникшим из-за рельефа вокруг глаз. Таким образом, лучше организовывать поиск угла только в изображении зрачка, найденном шагом ранее. Далее данные о координатах зрачка и угла передаются в модуль математики MathModule, реализованный специально для более точного подсчета координат и дальнейшего применения аппроксимации координат курсора.

Взаимодействие курсора и ОС

Так как нет возможности нажать на кнопки управления курсором, используем события наведения курсора на объект. При наведении курсора на кнопку панели управления запускается таймер с интервалом в 3000 мс (интервал выбран для обучения, в «боевом» режиме среднее 400мс-1с). По истечению данного таймера выбирается действие, которое будет совершать курсор и таймер вновь запускается для совершения действия.

Для более простого взаимодействия с пользователем создана форма *CustomCursorForm,* которая отображается при выборе действия и при наличии активного выбранного действия. Данная форма отображает *ProgressBar* под курсором мыши, дающий понять сколько времени осталось до выбора или совершения действия.

Для удобной отладки программы так же создан механизм логирования действий курсора, нажатых клавиш экранной клавиатуры и координат объекта на изображении, полученном с камеры.

Заключение

В данном проекте реализован прототип системы альтернативного управления ПК. Это доказывает возможность создания дешевого и работоспособного продукта для Российского и зарубежного рынка. Что бы с уверенностью заявлять о скорости и точности работы алгоритма, требуется собрать тестовые данные на основе нескольких популярных среди исследователей алгоритмах и сравнить их.

Следующим шагом в развитии продукта будет стороннее обучающее приложение, поскольку не каждый пользователь сможет быстро приспособиться к управлению курсором с помощью глаз.

Также планируется рассмотрение других алгоритмов или подключение нейронных сетей для изучения поведения пользователя и, возможно, подстройки системы к конкретному пользователю во время работы.

Список литературы

1. А.Л. Ярбус Академия наук СССР «Роль движений глаз в процессе зрения». Москва 1996г.
2. Jacob, R. J. K. Eye tracking in advanced interface design, 1995.
3. Klingenhoefer, S., and Bremmer, F. Saccadic suppression of displacement in face of saccade adaptation. Vision Research 51, 8 (2011), 881 – 889. Perception and Action: Part II.
4. Гиппенрейтер Ю. Б. Движения человеческого глаза / Ю. Б. Гиппенрейтер.
5. Grossberg, S., Srihasam, K., and Bullock, D. Neural dynamics of saccadic and smooth pursuit eye movement coordination during visual tracking of unpredictably moving targets. Neural Networks 27, 0 (2012), 1 – 20
6. Mouse cursor control with head and eye movements: A low-cost approach by Yat-Sing Yeung, BSc. 1220 Vienna, 2012
7. Gao, D., Yin, G., Cheng, W., and Feng, X. Non-invasive eye tracking technology based on corneal reflex. Procedia Engineering 29, 0 (2012), 3608 – 3612. 2012 International Workshop on Information and Electronics Engineering.
8. P. Viola and M. Jones. Robust real-time face detection. IJCV 57(2), 2004
9. Lienhart R., Kuranov E., Pisarevsky V.: Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In: PRS 2003, pp. 297-304 (2003)
10. Huang, Y., Wang, Z., and Ping, A. Non-contact gaze tracking with head movement adaptation based on single camera, 2009.
11. Sewell, W., and Komogortsev, O. Real-time eye gaze tracking with an unmodified commodity webcam employing a neural network. In Proceedings of the 28th of the international conference extended

abstracts on Human factors in computing systems. New York, NY, USA, 2010.

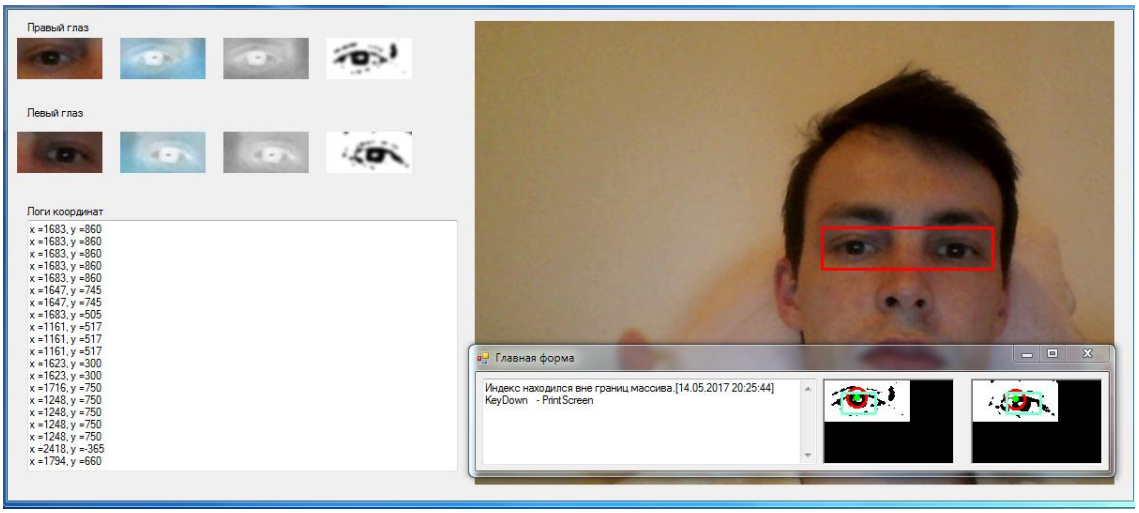
12. Torricelli, D., Conforto, S., Schmid, M., and DAlessio, T. A neuralbased remote eye gaze tracker under natural head motion. Computer Methods and Programs in Biomedicine 92, 1 (2008), 66 – 78.
13. Соловьев С.М., Разработка системы альтернативного управления компьютером. Управление цветным объектом. г. Ярославль, 2016г.
14. @Llammt, «Бинаризация изображений: алгоритм Брэдли», 2016. https://habrahabr.ru/post/278435/
15. Derek Bradley , Gerhard Roth Adaptive thresholding using the integral image, 2007.
16. Gaze Tracking Using A Regular Web Camera, Grahamstown, South Africa, 2012.
17. Документация по AForge.Net http://www.aforgenet.com/
18. Портал разработчиков Microsoft https://msdn.microsoft.com/

Приложение

Листинг программы

MainFormControl

Основная форма приложения, обеспечивает взаимосвязь обработчика видео и курсор ОС.




```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using gma.System.Windows;
using System.Runtime.InteropServices;


namespace EyeTrackerProject
{
    public partial class MainFormControl : Form
    {
        public int timeInterval { set { } get { return 3000; } }
        UserActivityHook actHook;
        CustomCursorForm mouse_form;
        int timing;
        CustomMouse _CM;
        Timer _timer, _progress_timer;
        public string MOUSEEVENTACTION = "LeftClick";
        ControlPanelForm _controlsForm;
        bool ActionIsEnd; //Действие совершено
        EyeVisionController VCForm;
        public MainFormControl()
        {
            TopMost = true;
            this.Hide();
            _CM = new CustomMouse();
            _controlsForm = new ControlPanelForm();
```

```
            _controlsForm.Owner = this;
            _controlsForm.setOwner();
            _timer = new Timer();
            _progress_timer = new Timer();
            _progress_timer.Interval = (int)timeInterval / 12;
            _progress_timer.Tick += PrintProgress;
            _progress_timer.Enabled = true;
            InitializeComponent();
            mouse_form = new CustomCursorForm();
            mouse_form.Owner = this;
            VCForm = new EyeVisionController();
            VCForm.Owner = this;
            VCForm.Show();
            // mouse_form.Show();
        }
        [DllImport("user32.dll", CharSet = CharSet.Auto, CallingConvention =
CallingConvention.StdCall)]
        public static extern void mouse_event(int dwFlags, int dx, int dy, int cButtons,
int dwExtraInfo);

        [DllImport("user32.dll", SetLastError = true)]
        static extern bool SetCursorPos(int X, int Y);

        public void MouseMoved(object sender, MouseEventArgs e)
        {

        }

        public void setMouseCoordinates(int x, int y)
        {
            SetCursorPos(x, y);
            Point tempPoint = new Point(x, y + 15);
            mouse_form.DesktopLocation = tempPoint;
            if (_CM.MouseMoved(x, y))
            {
                refreshEventHandlers();
                // LogWrite("Обновили таймеры");
            }
            if (y < _controlsForm.Height && x >
(SystemInformation.PrimaryMonitorSize.Width - _controlsForm.Width) / 2 && x <
(SystemInformation.PrimaryMonitorSize.Width + _controlsForm.Width) / 2)
            {
                if (!_controlsForm.IsShowed)
                {
                    _controlsForm.Show();
                    mouse_form.Show();
                    _controlsForm.IsShowed = true;
                    LogWrite("Показываем форму ");
                }
            }
            else
            {
                if (_controlsForm.IsShowed)
                {
                    _controlsForm.Hide();
                    _controlsForm.IsShowed = false;
                    LogWrite("Скрываем форму ");
                }
            }
        }

        public void MyKeyDown(object sender, KeyEventArgs e)
        {
            LogWrite("KeyDown   - " + e.KeyData.ToString());
```

```
        }
        enum MouseEvent
        {
            MOUSEEVENTF_LEFTDOWN = 0x02,
            MOUSEEVENTF_LEFTUP = 0x04,
            MOUSEEVENTF_RIGHTDOWN = 0x08,
            MOUSEEVENTF_RIGHTUP = 0x10,
        }
        public void MyKeyPress(object sender, KeyPressEventArgs e)
        {
            LogWrite("KeyPress  - " + e.KeyChar);
        }

        public void MyKeyUp(object sender, KeyEventArgs e)
        {
            LogWrite("KeyUp         - " + e.KeyData.ToString());
        }

        public void LogWrite(string txt)
        {
            logger_tb.AppendText("[" + DateTime.Now.ToString() + "] " + txt +
Environment.NewLine);
            logger_tb.SelectionStart = logger_tb.Text.Length;
        }

        private void MainFormControl_Load(object sender, EventArgs e)
        {
            actHook = new UserActivityHook(); // crate an instance with global hooks
            // hang on events
            actHook.OnMouseActivity += new MouseEventHandler(MouseMoved);
            actHook.KeyDown += new KeyEventHandler(MyKeyDown);
            actHook.KeyPress += new KeyPressEventHandler(MyKeyPress);
            actHook.KeyUp += new KeyEventHandler(MyKeyUp);

            actHook.Start();
        }

        private void DoClick(object sender, EventArgs e)
        {
            _CM.DoClick();
            setActionEnded(sender, e);
        }

        private void DoDBLClick(object sender, EventArgs e)
        {
            _CM.DoDBLClick();
            setActionEnded(sender, e);
        }

        private void DoRightClick(object sender, EventArgs e)
        {
            _CM.DoRightClick();
            _timer.Tick -= DoRightClick;
            _timer.Tick += DoClick;
        }

        private void DoMove(object sender, EventArgs e)
        {
            _CM.DoMoveLeftMouse();
            _timer.Tick -= DoMove;
            _timer.Tick += DoMoveRightMouse;
        }

        private void DoMoveRightMouse(object sender, EventArgs e)
```

```
{

    _CM.DoMoveRightMouse();
    _timer.Tick -= DoMoveRightMouse;
    setActionEnded(sender, e);
}

private void PrintProgress(object sender, EventArgs e)
{
    mouse_form.PrintProgress();
}

/// <summary>
/// Скролл
/// </summary>
/// <param name="side">1 - Вверх, 0 - Вниз</param>
public void DoScroll(int side)
{

}

public void refreshEventHandlers()
{
    mouse_form.ClearProgress();
    if (ActionIsEnd)
    {
        ActionIsEnd = false;
        LogWrite("Остановили таймеры");
        mouse_form.ClearProgress();
        mouse_form.Hide();
        _progress_timer.Stop();
        MOUSEEVENTACTION = "";
        _timer.Stop();
        _timer.Tick -= DoClick;
        _timer.Tick -= DoDBLClick;
        _timer.Tick -= DoRightClick;
        _timer.Tick -= setActionEnded;
    }
    else
    {
        _timer.Stop();
        _timer.Start();
        mouse_form.ClearProgress();
    }
}

public void startMouseTimer()
{
    LogWrite("Запуск таймеров в главной форме с событием " + MOUSEEVENTACTION);
    _timer.Dispose();
    _timer = new Timer();
    _timer.Interval = timeInterval;
    switch (MOUSEEVENTACTION)
    {
        case "LeftClick":
            {
                _timer.Tick += DoClick;
            }
            break;
        case "DBLLeftClick":
            {
                _timer.Tick += DoDBLClick;
            }
            break;
```

```
                case "RightClick":
                    {
                        _timer.Tick += DoRightClick;
                    }
                    break;
                case "Move":
                    {
                        _timer.Tick += DoMove;
                    }
                    break;
            }
            _timer.Enabled = true;

            _timer.Start();
            mouse_form.ClearProgress();
            mouse_form.Show();
            _progress_timer.Start();
        }

        public void setActionEnded(object sender, EventArgs e)
        {
            ActionIsEnd = true;
            refreshEventHandlers();
        }

        public void printControlsForm()
        {

        }

        private void MainFormControl_FormClosed(object sender, FormClosedEventArgs e)
        {
            Application.Exit();
        }
    }
}
```

EyeVisionController

Форма на которой производится обработка изображения

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;
using AForge;
using AForge.Imaging.Filters;
using AForge.Imaging;
using AForge.Video;
using AForge.Video.DirectShow;
using AForge.Math.Geometry;
using Emgu.CV;
using Emgu.CV.Structure;
using Emgu.CV.CvEnum;
using System.IO;
using System.Diagnostics;

namespace EyeTrackerProject
{
    public partial class EyeVisionController : Form
    {
        Image<Bgr, Byte> currentFrame;
        Capture grabber;
        HaarCascade eye;
        public static UInt32[,] pixel;
        MCvFont font = new MCvFont(FONT.CV_FONT_HERSHEY_TRIPLEX, 0.5d, 0.5d);
        Image<Gray, byte> result = null;
        Image<Gray, byte> gray = null;
        List<Image<Gray, byte>> trainingImages = new List<Image<Gray, byte>>();
        List<string> labels = new List<string>();
        List<PictureBox> pictures = new List<System.Windows.Forms.PictureBox>();
        MathModule coords;
        MainFormControl owner_form;
        PointF[][] corners;
        public EyeVisionController()
        {
            InitializeComponent();
            //Создадим картинки для передачи изображения
            for (int i = 0; i < 8; i++)
            {
                int y = (int)(i / 4);
                pictures.Add(new System.Windows.Forms.PictureBox());
                pictures[i].Location = new System.Drawing.Point(10 + (i % 4) * 110, 30 +
y * 100);
                pictures[i].Name = "pictureBox" + i.ToString();
                pictures[i].Size = new System.Drawing.Size(100, 100);
                pictures[i].TabIndex = 10;
                pictures[i].TabStop = false;
                this.Controls.Add(pictures[i]);
            }
            coords = new MathModule(SystemInformation.PrimaryMonitorSize.Width * 41 /
(800 * 30), SystemInformation.PrimaryMonitorSize.Height);
            //Каскад хаара
            eye = new HaarCascade("haarcascade_mcs_eyepair_big.xml");
            //Камера
            grabber = new Capture();
            grabber.QueryFrame();
            //Подписка на событие
            Application.Idle += new EventHandler(FrameGrabber);
        }
```

```
        void FrameGrabber(object sender, EventArgs e)
        {
            Invert filterInvert = new Invert();

            IFilter filterGrayscale = Grayscale.CommonAlgorithms.RMY;
            Threshold th = new Threshold(240);
            BlobCounter bl;

            ExtractBiggestBlob fil2;

            currentFrame = grabber.QueryFrame().Resize(800, 600,
Emgu.CV.CvEnum.INTER.CV_INTER_CUBIC);
            gray = currentFrame.Convert<Gray, Byte>();
            MCvAvgComp[][] facesDetected = gray.DetectHaarCascade(
          eye,
          1.2,
          10,
          Emgu.CV.CvEnum.HAAR_DETECTION_TYPE.DO_CANNY_PRUNING,
          new Size(20, 20));
            Bitmap[] test = new Bitmap[5];
            int width = 0, height = 0;
            int kek = 0;
            UInt32 point = new UInt32();
            int t = 0;
            foreach (MCvAvgComp f in facesDetected[0])
            {
                t = t + 1;
                result = currentFrame.Copy(f.rect).Convert<Gray, byte>().Resize(100, 100,
Emgu.CV.CvEnum.INTER.CV_INTER_CUBIC);
                width = f.rect.Width / 2;
                height = f.rect.Height;
                test[0] = new Bitmap(width, height);
                test[1] = new Bitmap(width, height);
                for (var i = 0; i < width; i++)
                {
                    for (var j = 0; j < height; j++)
                    {
                        test[0].SetPixel(i, j, currentFrame.Bitmap.GetPixel(f.rect.X + i,
f.rect.Y + j));
                    }
                }
                for (var i = 0; i < width; i++)
                {
                    for (var j = 0; j < height; j++)
                    {

                        test[1].SetPixel(i, j, currentFrame.Bitmap.GetPixel(f.rect.X +
width + i, f.rect.Y + j));
                    }
                }
                currentFrame.Draw(f.rect, new Bgr(Color.Red), 2);
                kek++;
            }
            t = 0;
            imageBoxFrameGrabber.Image = currentFrame;
            //создадим фильтр
            EuclideanColorFiltering filter = new EuclideanColorFiltering();
            //Применим фильтр
            int x_l, y_l, x_r = 0, y_r = 0;
            for (var index = 0; index < test.Length; index++)
            {
                if (test[index] != null)
```

```
                {
                    pictures[index * 4].Image = test[index];
                    System.Drawing.Bitmap kek123 = test[index].Clone(new Rectangle(0, 0,
width, height), System.Drawing.Imaging.PixelFormat.Format24bppRgb);
                    kek123 = filterInvert.Apply((Bitmap)kek123);
                    AForge.Imaging.Image.FormatImage(ref kek123);
                    pictures[index * 4 + 1].Image = kek123; // Принт изображения
                    kek123 = filterGrayscale.Apply(kek123); // Наложим грейскейл
                    pictures[index * 4 + 2].Image = kek123; // Принт изображения
                    kek123 = BrightnessContrast.Bradley_threshold(kek123);

                    // РИСУЕМ КРУГИ
                    Image<Bgr, Byte> imgOriginal = new Image<Bgr, byte>(kek123);
                    Image<Gray, Byte> imgProcessed = (new Image<Gray, Byte>(kek123));


                    imgProcessed = imgProcessed.SmoothGaussian(9);

                    CircleF[] circles = imgProcessed.HoughCircles(new Gray(70), new
Gray(20), 1.5, imgProcessed.Height, 3, 12)[0];
                    foreach (CircleF circle in circles)
                    {
                        CvInvoke.cvCircle(imgOriginal, new
System.Drawing.Point((int)circle.Center.X, (int)circle.Center.Y), 3, new MCvScalar(0,
255, 0), -1, LINE_TYPE.CV_AA, 0);
                        if (index == 0)
                        {
                            x_r = (int)circle.Center.X;
                            y_r = (int)circle.Center.Y;
                        }
                        else
                        {
                            x_l = (int)circle.Center.X;
                            y_l = (int)circle.Center.Y;
                            coords.addPoint(x_l, y_l, x_r, y_r);
                        }


                        imgOriginal.Draw(circle, new Bgr(Color.Red), 2);

                        if (textBox1.Text != "")
textBox1.AppendText(Environment.NewLine);
                        textBox1.AppendText("x =" + coords.getX() + ", y =" +
coords.getY());
                        textBox1.ScrollToCaret();
                        break;
                    }
                    try
                    {
                        int cornerCount = 2;
                        corners = setCornerRoi(imgOriginal,
index.Equals(0)).GoodFeaturesToTrack(3, 0.8, imgOriginal.Width, 3);

                        if (index.Equals(0))
                        {
                            coords.setLeftCorner(corners[0][0].X, corners[0][0].Y);
                            owner_form.left_eye.Image = imgOriginal.ToBitmap();
                        }
                        else
                        {
                            coords.setRightCorner(corners[0][0].X, corners[0][0].Y);
                            owner_form.right_eye.Image = imgOriginal.ToBitmap();
                        }
                        owner_form.setMouseCoordinates(coords.getX(), coords.getY());
```

```
                    }
                    catch (Exception except)
                    {
                        owner_form.logger_tb.Text = except.Message;
                    }
                    pictures[index * 4 + 3].Image = imgProcessed.ToBitmap();
                }
            }
        }

        private Emgu.CV.Image<Emgu.CV.Structure.Bgr, byte>
setCornerRoi(Emgu.CV.Image<Emgu.CV.Structure.Bgr, byte>  imgOriginal, bool left)
        {
            Emgu.CV.Image<Emgu.CV.Structure.Bgr, byte> img = imgOriginal.Clone();
            int x, y, width, height;
            if (left)
            {
                x = (int)(img.Width * 0.2);
                y = (int) (img.Height * 0.3);
            }
            else
            {
                x = (int)(img.Width * 0.4);
                y = (int)(img.Height * 0.3);
            }
            width = (int)(img.Width * 0.4);
            height = (int)(img.Height * 0.5);
            img.ROI = new Rectangle(x, y, width, height);
            imgOriginal.Draw(new Rectangle(x, y, width, height), new
Bgr(Color.Aquamarine), 2);
            return img;
        }


        private void button1_Click(object sender, EventArgs e)
        {
            FrameGrabber(sender, e);
        }

        private void EyeVisionController_Load(object sender, EventArgs e)
        {
            owner_form = (MainFormControl)this.Owner as MainFormControl;

        }

    }
}
```

MathModule

Класс для математических операций (вычисление координат).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EyeTrackerProject
{
    class MathModule
    {
        public List<int> x_left;
        public List<int> x_right;
        public List<int> y_left;
        public List<int> y_right;

        public float left_corner_x, left_corner_y;
        public float right_corner_x, right_corner_y;

        public List<float> x_mouse;
        public List<float> y_mouse;

        public float width_ratio;
        public float height_ratio;
        public MathModule()
        {
            x_left = new List<int>();
            x_right = new List<int>();
            y_left = new List<int>();
            y_right = new List<int>();
            x_mouse = new List<float>();
            y_mouse = new List<float>();
        }

        public MathModule(float w, float h):this()
        {
            width_ratio = w;
            height_ratio = h;
        }
        public void addPoint(int x_l, int y_l, int x_r, int y_r)
        {
            x_left.Add(x_l);
            y_left.Add(y_l);
            x_right.Add(x_r);
            y_right.Add(y_r);
            ApproximateCoordinates();
        }

        public void ApproximateCoordinates()
        {
            float x, y;
            x = x_left.Last() * width_ratio * left_corner_x;
            x += x_right.Last() * width_ratio * right_corner_x;
            x_mouse.Add(x / 2);

            y = y_left.Last() * left_corner_y;
            y += y_right.Last() * right_corner_y;
            y_mouse.Add( height_ratio - (y / 2));
        }

        public void setLeftCorner(float x, float y)
```

```
        {
            left_corner_x = x;
            left_corner_y = y  * 5;
        }

        public void setRightCorner(float x, float y)
        {
            right_corner_x = x;
            right_corner_y = y * 5;
        }

        public int getX()
        {
            if (x_mouse.Count() > 0)
            {
                return (int) x_mouse.Last();
            }
            return 0;
        }

        public int getY()
        {
            if (x_mouse.Count() > 0)
            {
                return (int) y_mouse.Last();
            }
            return 0;
        }
    }
}
```

BrightnessContrast

Вспомогательный класс для обработки изображения.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using AForge.Imaging.Filters;
using Emgu.CV;
using Emgu.CV.Structure;

namespace EyeTrackerProject
{
    class BrightnessContrast
    {
        //якрость
        public static UInt32 Brightness(UInt32 point, int poz, int lenght)
        {
            int R;
            int G;
            int B;

            int N = (100 / lenght) * poz; //кол-во процентов

            R = (int)(((point & 0x00FF0000) >> 16) + N * 128 / 100);
            G = (int)(((point & 0x0000FF00) >> 8) + N * 128 / 100);
            B = (int)((point & 0x000000FF) + N * 128 / 100);

            //контролируем переполнение переменных
            if (R < 0) R = 0;
            if (R > 255) R = 255;
            if (G < 0) G = 0;
            if (G > 255) G = 255;
            if (B < 0) B = 0;
            if (B > 255) B = 255;

            point = 0xFF000000 | ((UInt32)R << 16) | ((UInt32)G << 8) | ((UInt32)B);

            return point;
        }

        //контрастность
        public static UInt32 Contrast(UInt32 point, int poz, int lenght)
        {
            int R;
            int G;
            int B;

            int N = (100 / lenght) * poz; //кол-во процентов

            if (N >= 0)
            {
                if (N == 100) N = 99;
                R = (int)((((point & 0x00FF0000) >> 16) * 100 - 128 * N) / (100 - N));
                G = (int)((((point & 0x0000FF00) >> 8) * 100 - 128 * N) / (100 - N));
                B = (int)(((point & 0x000000FF) * 100 - 128 * N) / (100 - N));
            }
            else
            {
                R = (int)((((point & 0x00FF0000) >> 16) * (100 - (-N)) + 128 * (-N)) /
100);
```

```
                G = (int)((((point & 0x0000FF00) >> 8) * (100 - (-N)) + 128 * (-N)) /
100);
                B = (int)(((point & 0x000000FF) * (100 - (-N)) + 128 * (-N)) / 100);
            }

            //контролируем переполнение переменных
            if (R < 0) R = 0;
            if (R > 255) R = 255;
            if (G < 0) G = 0;
            if (G > 255) G = 255;
            if (B < 0) B = 0;
            if (B > 255) B = 255;

            point = 0xFF000000 | ((UInt32)R << 16) | ((UInt32)G << 8) | ((UInt32)B);

            return point;
        }

        //бинаризация Бредли
        public static Bitmap Bradley_threshold(Bitmap Img)
        {
            Bitmap result = new Bitmap(Img);
            int width = Img.Width;
            int height = Img.Height;
            int S = (int)(width / 8);
            int s2 = S / 2;
            double t = 0.08;
            Int32[] integral_image;
            Int32 sum = 0;
            int count = 0;
            int index;
            int x1, y1, x2, y2;

            //рассчитываем интегральное изображение
            integral_image = new Int32[width * height];

            for (int i = 0; i < width; i++)
            {
                sum = 0;
                for (int j = 0; j < height; j++)
                {
                    index = j * width + i;
                    sum += Img.GetPixel(i, j).ToArgb();
                    if (i == 0)
                        integral_image[index] = sum;
                    else
                        integral_image[index] = integral_image[index - 1] + sum;
                }
            }

            //находим границы для локальные областей
            for (int i = 0; i < width; i++)
            {
                for (int j = 0; j < height; j++)
                {
                    index = j * width + i;

                    x1 = i - s2;
                    x2 = i + s2;
                    y1 = j - s2;
                    y2 = j + s2;

                    if (x1 < 0)
                        x1 = 0;
```

```
                    if (x2 >= width)
                        x2 = width - 1;
                    if (y1 < 0)
                        y1 = 0;
                    if (y2 >= height)
                        y2 = height - 1;

                    count = (x2 - x1) * (y2 - y1);

                    sum = integral_image[y2 * width + x2] - integral_image[y1 * width +
x2] -
                                integral_image[y2 * width + x1] + integral_image[y1 *
width + x1];
                    if ((long)(Img.GetPixel(i, j).ToArgb() * count) < (long)(sum * (1.0 -
t)))
                        result.SetPixel(i, j, Color.White);
                    else
                        result.SetPixel(i, j, Color.Black);
                }
            }

            return result;
        }
        public static Bitmap InvertImageColorMatrix(Image originalImg)
        {
            float[][] sepiaValues = {
                new float[]{.393f, .349f, .272f, 0, 0},
                new float[]{.769f, .686f, .534f, 0, 0},
                new float[]{.189f, .168f, .131f, 0, 0},
                new float[]{0, 0, 0, 1, 0},
                new float[]{0, 0, 0, 0, 1}
            };
            //float[][] sepiaValues = {
            //  new float[] { 0, 0, 0, 0, 0},
            //  new float[] { 0, 0, 0, 0, 0},
            //  new float[] { 0, 0, 0, 0, 0},
            //  new float[] { 0, 0, 0, 0.25f, 0},
            //  new float[] { 0, 0, 0, 0, 0},
            //};
            System.Drawing.Imaging.ColorMatrix sepiaMatrix = new
System.Drawing.Imaging.ColorMatrix(sepiaValues);
            System.Drawing.Imaging.ImageAttributes IA = new
System.Drawing.Imaging.ImageAttributes();
            IA.SetColorMatrix(sepiaMatrix);
            Bitmap sepiaEffect = (Bitmap)originalImg.Clone();
            using (Graphics G = Graphics.FromImage(sepiaEffect))
            {
                G.DrawImage(originalImg, new Rectangle(0, 0, sepiaEffect.Width,
sepiaEffect.Height), 0, 0, sepiaEffect.Width, sepiaEffect.Height, GraphicsUnit.Pixel,
IA);
            }
            return sepiaEffect;
        }

        public static Bitmap HistogramEqualization(Bitmap Img)
        {
            // create filter
            HistogramEqualization filter = new HistogramEqualization();
            // process image
            filter.ApplyInPlace(Img);
            return Img;
        }

        public static Image<Bgr, Byte> ShowCircles(Image<Gray, Byte> Img_Source_Gray)
```

```
        {
            Image<Bgr, byte> Img_Result_Bgr = new Image<Bgr, byte>(Img_Source_Gray.Width,
Img_Source_Gray.Height);
            CvInvoke.cvCvtColor(Img_Source_Gray.Ptr, Img_Result_Bgr.Ptr,
Emgu.CV.CvEnum.COLOR_CONVERSION.CV_GRAY2BGR);
            Gray cannyThreshold = new Gray(10);
            Gray circleAccumulatorThreshold = new Gray(16);
            double Resolution = 5.90;
            double MinDistance = 10.0;
            int MinRadius = 0;
            int MaxRadius = 0;

            CircleF[] HoughCircles = Img_Source_Gray.Clone().HoughCircles(
                                    cannyThreshold,
                                    circleAccumulatorThreshold,
                                    Resolution, //Resolution of the accumulator used to
detect centers of the circles
                                    MinDistance, //min distance
                                    MinRadius, //min radius
                                    MaxRadius //max radius
                                    )[0]; //Get the circles from the first channel
            #region draw circles
            foreach (CircleF circle in HoughCircles)
                Img_Result_Bgr.Draw(circle, new Bgr(Color.Red), 1);
            #endregion

            return Img_Result_Bgr;
        }
        public static Bitmap CropImage(Bitmap source, Rectangle section)
        {
            Bitmap bmp = new Bitmap(section.Width, section.Height);
            Graphics g = Graphics.FromImage(bmp);
            g.DrawImage(source, 0, 0, section, GraphicsUnit.Pixel);
            return bmp;
        }
    }
}
```

ControlPanelForm

Форма для выбора действия. Отображается поверх всех окон при наведении мыши в верхнюю часть экрана. Для выбора действия нужно задержать курсор на 3 сек.

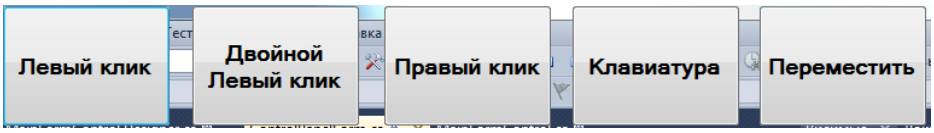


Рис 4.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;
using System.IO;
using System.Runtime.InteropServices;


namespace EyeWindowsController
{
    public partial class ControlPanelForm : Form
    {
        int HEIGHT = SystemInformation.PrimaryMonitorSize.Height / 10;
        int WIDTH = SystemInformation.PrimaryMonitorSize.Width/ 2;
        public bool IsShowed = false;
        public ControlPanelForm()
        {
            InitializeComponent();
            this.Width = WIDTH;
            this.Height = Height;
            this.Left = SystemInformation.PrimaryMonitorSize.Width / 2 - WIDTH / 2;
            this.Top = 0;
            this.FormBorderStyle = FormBorderStyle.None;
            this.AllowTransparency = true;
            this.BackColor = Color.AliceBlue;//цвет фона
            this.TransparencyKey = this.BackColor;//он же будет заменен на прозрачный
цвет

        }
        MainFormControl main;
        Timer _timer;
        bool ActionIsSelected;

        public void setOwner()
        {
            main = (MainFormControl)this.Owner as MainFormControl;
        }
        private void leftMouseClickBTN_MouseEnter(object sender, EventArgs e)
        {
            SetClick();
        }

        public void refreshTimer()
        {
```

```
    if (!ActionIsSelected)
    {
        main.MOUSEEVENTACTION = "";
        main.setActionEnded(null, null);
    }
}

private void DblLeftMouseClickBTN_MouseEnter(object sender, EventArgs e)
{
    SetClick();
}

private void DblLeftMouseClickBTN_MouseLeave(object sender, EventArgs e)
{
    refreshTimer();
}

private void leftMouseClickBTN_MouseLeave(object sender, EventArgs e)
{
    refreshTimer();
}

private void RightMouseClickBTN_MouseLeave(object sender, EventArgs e)
{
    refreshTimer();
}

private void keyboard_btn_MouseLeave(object sender, EventArgs e)
{
    refreshTimer();
}

private void move_btn_MouseLeave(object sender, EventArgs e)
{
    refreshTimer();
}

private void SetClick()
{
    main.MOUSEEVENTACTION = "LeftClick";
    main.startMouseTimer();
    ActionIsSelected = false;
}

private void leftMouseClickBTN_Click(object sender, EventArgs e)
{
    ActionIsSelected = true;
    main.MOUSEEVENTACTION = "LeftClick";
    main.startMouseTimer();
    this.Hide();
}

private void DblLeftMouseClickBTN_Click(object sender, EventArgs e)
{
    ActionIsSelected = true;
    main.MOUSEEVENTACTION = "DBLLeftClick";
    main.startMouseTimer();
    this.Hide();
}

private void RightMouseClickBTN_Click(object sender, EventArgs e)
{
    ActionIsSelected = true;
    main.MOUSEEVENTACTION = "RightClick";
```

```
            main.startMouseTimer();
            this.Hide();
        }

        private void keyboard_btn_Click(object sender, EventArgs e)
        {
            try
            {
                if (!Process.GetProcessesByName("osk").Any())
                {
                    var path64 = @"C:\Windows\winsxs\amd64_microsoft-windows-
osk_31bf3856ad364e35_6.1.7600.16385_none_06b1c513739fb828\osk.exe";
                    var path32 = @"C:\windows\system32\osk.exe";
                    var path = (Environment.Is64BitOperatingSystem) ? path64 : path32;
                    Process.Start(path);
                }
            }
            catch (Exception exc)
            {
                main.LogWrite("Ошибка запуска Клавиатуры " + exc);
            }
        }

        private void move_btn_Click(object sender, EventArgs e)
        {
            ActionIsSelected = true;
            main.MOUSEEVENTACTION = "Move";
            main.startMouseTimer();
            this.Hide();
        }

        private void ControlPanelForm_Load(object sender, EventArgs e)
        {

        }
    }
}
```