Управление воспроизведением музыкальных композиций с помощью жестов.

Автор работы:

Михайлов Владимир Евгеньевич
студент департамента программной инженерии
факультета компьютерных наук
НИУ ВШЭ

Научный руководитель:

Сибирцева Елена Алексеевна
преподаватель НИУ ВШЭ,
победитель Microsoft Imagine Cup 2013 (российский этап)

Краткая аннотация.

В данной статье я хотел бы в доступной форме изложить некоторые базовые подходы, используемые в таком разделе машинного обучения, как «Компьютерное зрение».

Скорее всего, изложенный материал будет интересен тем, кто только начинает знакомиться с этим перспективным направлением в области информационных технологий.

Будут описаны некоторые распространенные алгоритмы обработки изображений и распознавания, которые (на взгляд автора) не являются сложными для понимания, но дают в какой-то степени интересный результат при правильной их компоновке.

Описание методов будет происходить на примере решения задачи «Управление воспроизведением музыкальных композиций с помощью жестов».

Полученные результаты имеют по большей части научно-образовательный характер. Автор осознает, что использованный подход может быть не оптимальным для решения поставленной задачи. Конструктивная критика и советы исключительно приветствуются.

Общая схема работы программы.

Практически любое решение задачи компьютерного зрения строится по следующей схеме: изображение получается на вход, некоторым образом обрабатывается, причесывается, затем строится некоторая математическая модель, анализ значений которой позволяет сделать необходимые выводы для решения поставленной задачи.

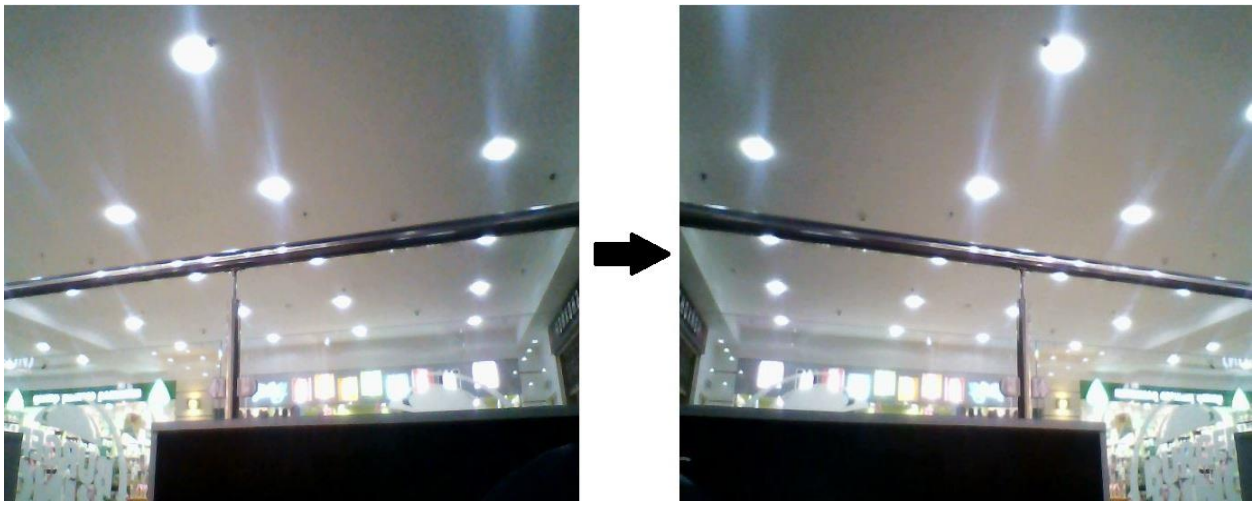
В нашем случае мы будем действовать по этой же «классической» схеме. На вход программе (алгоритму) будем подавать видео-поток с веб-камеры, из которого перед непосредственной демонстрацией жестов пользователь должен сохранить фон (изображение, на фоне которого он будет управлять воспроизведением композиций).

Обработка изображения.

Отражение.

При получении изображения с веб-камеры зачастую api предоставляет нам отраженное относительно вертикальной центральной оси изображение. Следовательно, для понятной и удобной работы необходимо отразить его обратно. Так будем поступать со всеми изображениями, поступающими к нам на вход.

$$Image(i,j) \rightarrow Image(i,n-j-1)$$



Результат отражения изображения

Размытие по Гауссу.

Фильтр размытия по Гауссу (“Gaussian blur”) – широко используемый в задачах компьютерного зрения метод для подавления шумов на изображении.

Гауссово размытие с радиусом r считается по формуле:

$$y(m,n) = \frac{1}{2\pi r^2} \sum_{u,v} e^{\frac{-(u^2+v^2)}{2r^2}} x(m+u, n+v)$$

То есть каждый пиксель исходного изображения переходит в пиксель результирующего изображения, равный сумме произведений некоторых соседних пикселей исходного изображения с некоторыми коэффициентами меньшими единицы.

Подобная операция называется «сверткой». Ядро свертки – матрица коэффициентов, на которые умножаются значения пикселей соседей просматриваемого пикселя в исходном изображении. Числа в матрице ядра свертки имеют нормальное распределение в трехмерном пространстве.

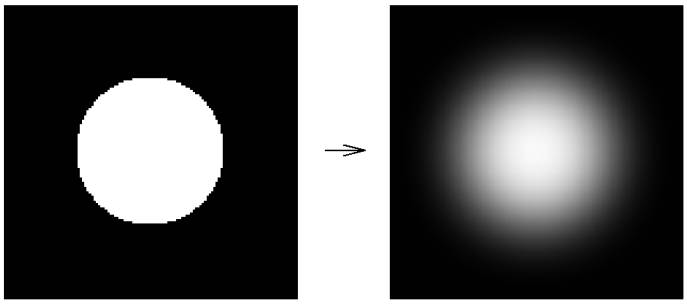
Вот как в простом случае может выглядеть реализация алгоритма размытия по Гауссу для ядра 3x3 на языке C# :

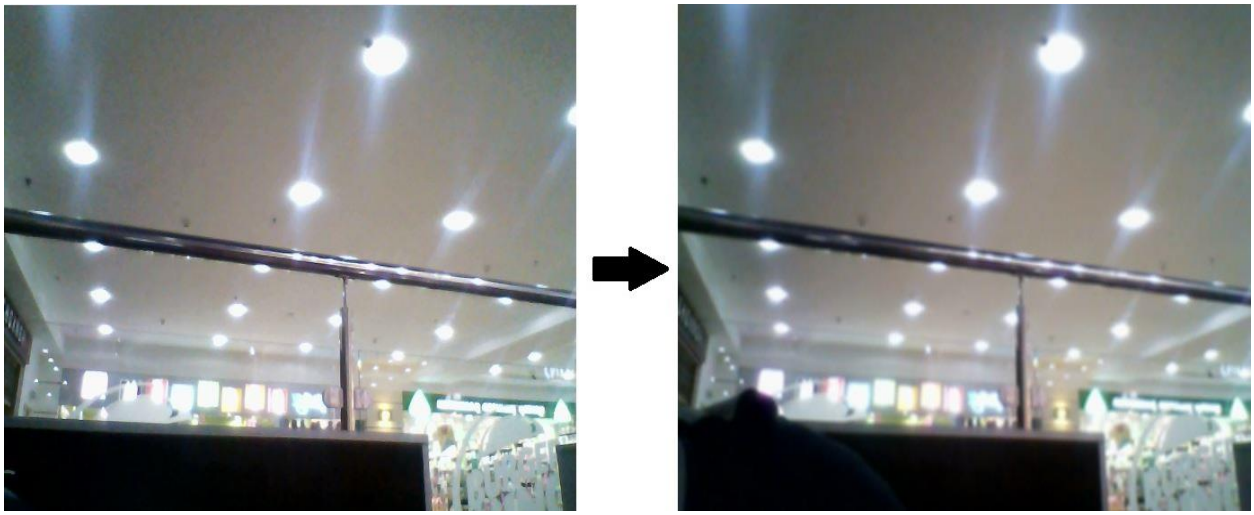
```
double[] gaussCore = new double[9] { 1d / 16, 1d / 8, 1d / 16,
                                     1d / 8, 1d / 4, 1d / 8,
                                     1d / 16, 1d / 8, 1d / 16 };

double[] core = gaussCore;

for (int i = 0; i < Height; i++) {
                for (int j = 0; j < Width; j++) {
                    convolutedMatrixImage[i][j] = 0;        // Инициализация
                    int coreInd = 0;                        // Индекс пробега по ядру

                    for (int zi = Math.Max(i - 1, 0); zi <= Math.Min(i + 1, Height - 1);
zi++) {
                        for (int zj = Math.Max(j - 1, 0); zj <= Math.Min(j + 1, Width -
1); zj++) {
                            convolutedMatrixImage[i][j] += originalMatrixImage[zi][zj] *
core[coreInd++];
                        }
                    }
                }
            }
```





Результат применения размытия по Гауссу

Разность между изображениями.

Как уже говорилось выше, в нашем подходе мы делаем допущение (вводим правило), что пользователь должен использовать нашу программу на фоне, который является статическим, который необходимо зафиксировать (нажатием на клавишу Enter, например) перед началом демонстрации жестов. Зная фон, мы можем получить маску динамического объекта, взяв разность между изображениями. Новые (динамические) объекты обозначим белым цветом, а фон закрасим черным.

Здесь полезно посмотреть на изменение пикселя по каждому каналу **RGB** и сказать, что изменение по модулю больше какого-то порога (зависит от качества изображения и освещения) означает появление нового объекта на изображении.

Также полезно рассмотреть наше изображение в цветовом пространстве **YC_BC_R**, в котором **Y'** – компонента яркости, а **C_B** и **C_R** являются синей и красной цветоразностными компонентами. Теоретически работа с данной моделью должна быть нечувствительна к свету: нам достаточно следить, изменились ли цветоразностные компоненты (берется минимальный порог равный 1-2) – но на практике результаты иногда получаются хуже (но иногда и значительно лучше), чем при работе с пространством **RGB**. Все зависит от освещения.

$$
\begin{aligned}
Y' &= 0 + (0.299 \cdot R'_D) + (0.587 \cdot G'_D) + (0.114 \cdot B'_D) \\
C_B &= 128 - (0.168736 \cdot R'_D) - (0.331264 \cdot G'_D) + (0.5 \cdot B'_D) \\
C_R &= 128 + (0.5 \cdot R'_D) - (0.418688 \cdot G'_D) - (0.081312 \cdot B'_D)
\end{aligned}
$$

Формулы преобразования цветового пространства RGB в YCC

Получение маски динамических объектов на изображении

Прим. Вообще получение маски из изображения не является наилучшим методом ввиду зашумления из-за низкого качества получаемого с веб-камеры изображения. Такие технологии, как Kinect используют инфракрасное (тепловое) излучение для получения маски человека и распознавания его жестов.

Морфологические операции.

Так называемые морфологические операции также имеют широкое применение в задачах компьютерного зрения, требующих подавления шума на изображении, но основной их задачей является устранение некоторого рода дефектов на изображении (лишние выпуклости и дыры).

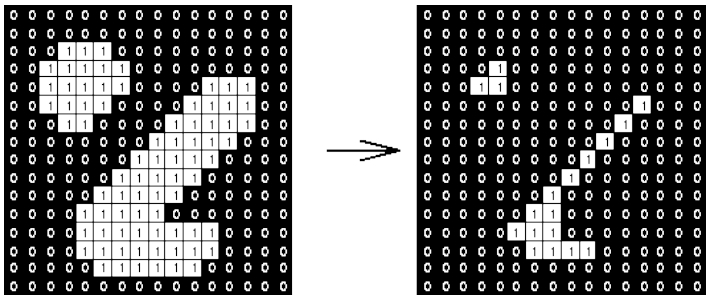
«Сужение».

Первая рассматриваемая морфологическая операция – «Сужение» (“Erosion”).

При применении морфологической операции сужения в простом случае (с рассматриваемым нами ядром размера 3x3) белый пиксель исходного изображения переходит в белый пиксель результирующего изображения тогда и только тогда, когда все его соседи в исходном изображении тоже белые, иначе он переходит в пиксель черного цвета. В более сложном случае применяются ядра большего размера, принцип остается прежним, но могут рассматриваться не все соседи. Также стоит отметить, что на практике, как правило, алгоритмы морфологии применяются несколько раз (например, 4 итерации) для достижения необходимого результата.

Реализация простого случая морфологической операции «Сужение» на языке C# :

```
for (int i = 0; i < Height; i++) {
                for (int j = 0; j < Width; j++) {
                    bool flag = true;
                    for (int zi = Math.Max(i - 1, 0); zi <= Math.Min(i + 1, Height - 1);
zi++) {
                        for (int zj = Math.Max(j - 1, 0); zj <= Math.Min(j + 1, Width -
1); zj++) {
                            if ((!(zi == i && zj == j)) && (matrixImage[zi][zj] == 0)) {
                                flag = false;
                            }
                        }
                    }
                    if (flag) {
                        filteredMatrixImage[i][j] = 255;
                    }
                    else {
                        filteredMatrixImage[i][j] = 0;
                    }
                }
            }
```

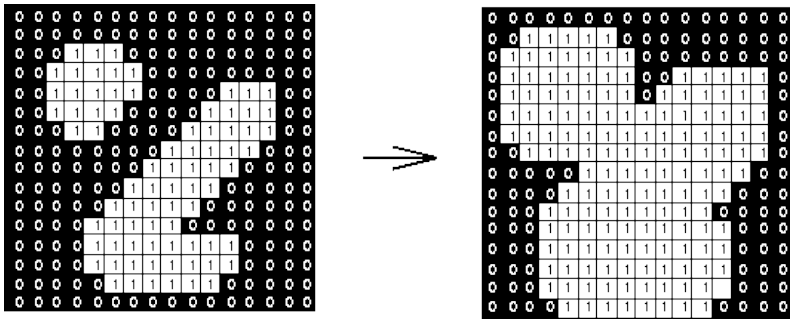


Результат применения операции «Сужение»

«Расширение».

Вторая рассматриваемая морфологическая операция – «Расширение» (“Dilation”).

При применении морфологической операции расширения (с рассматриваемым нами ядром размера 3x3) черный пиксель исходного изображения переходит в черный тогда и только тогда, когда все его соседи в исходном изображении имеют черный цвет, иначе он переходит в пиксель белого цвета. Так же, как и у операции «Сужение», при увеличении размера ядра меняются правила, по которым рассматриваются те или иные соседи.



Результат применения операции «Расширение»

«Открытие».

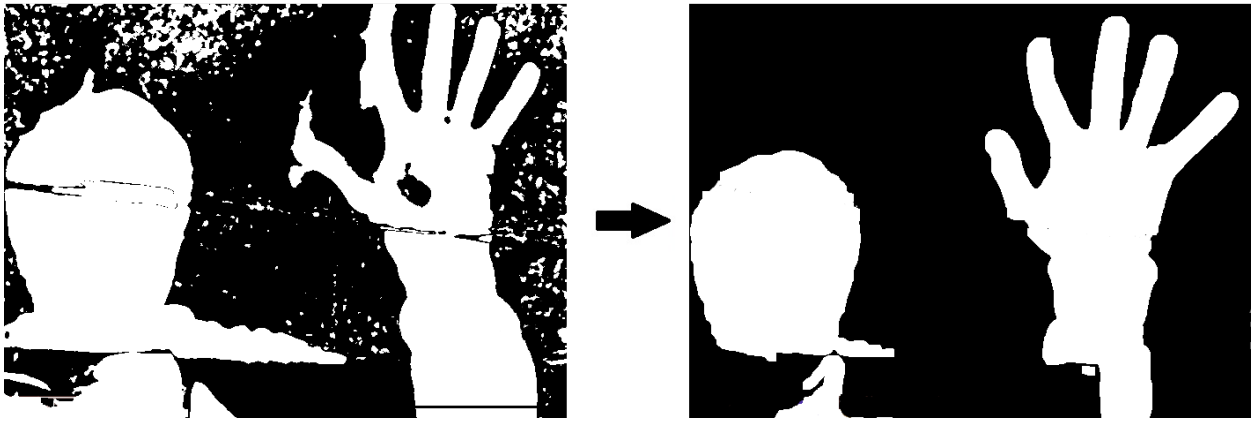
Морфологическая операция «Открытие» является последовательным применением морфологических операций «Сужение» и «Расширение» (например, 4 итерации одного метода, затем 4 итерации другого), применяемая в случае необходимости удаления шума в виде отдельных белых пикселей («соль»), либо мелких кластеров белых пикселей, либо для удаления тонких линий, находящихся в преимущественно черных зонах изображения. Стоит применять операцию с умом, чтобы не потерять значимые объекты (тонкие пальцы).

Результат применения операции открытия к изображению с сильным шумом

«Закрытие».

Морфологическая операция «Закрытие» является последовательным применением морфологических операций «Расширение» и «Сужение», применяемая в случае необходимости удаления дефектов объектов в виде отдельных черных пикселей («перец»), либо мелких кластеров черных пикселей, находящихся в преимущественно белых зонах изображения. Необходимо «знать меру», чтобы какие-нибудь объекты (пальцы) не слились в один.

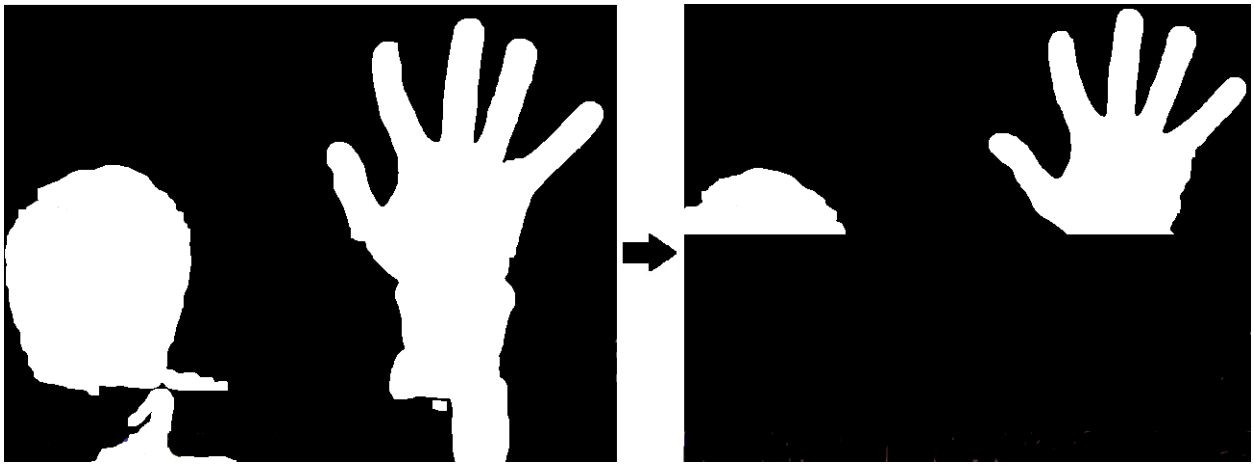
Результат применения операции закрытия к изображению с дефектами объектов



Результат применения морфологических операций

Удаление нижней части изображения.

Сделаем допущение (установим правило), что пользователь может демонстрировать жест, который должен быть распознан программой, только в верхней части экрана. Просто зальем черным цветом нижнюю половину изображения. Таким образом, мы оставляем пользователю пространство, в котором он может двигать руками (например, танцевать под проигрываемую музыку) без риска, что какие-то случайно продемонстрированные жесты будут приняты алгоритмом. Для себя же мы сокращаем объем шума, который, к сожалению, все равно может появляться.

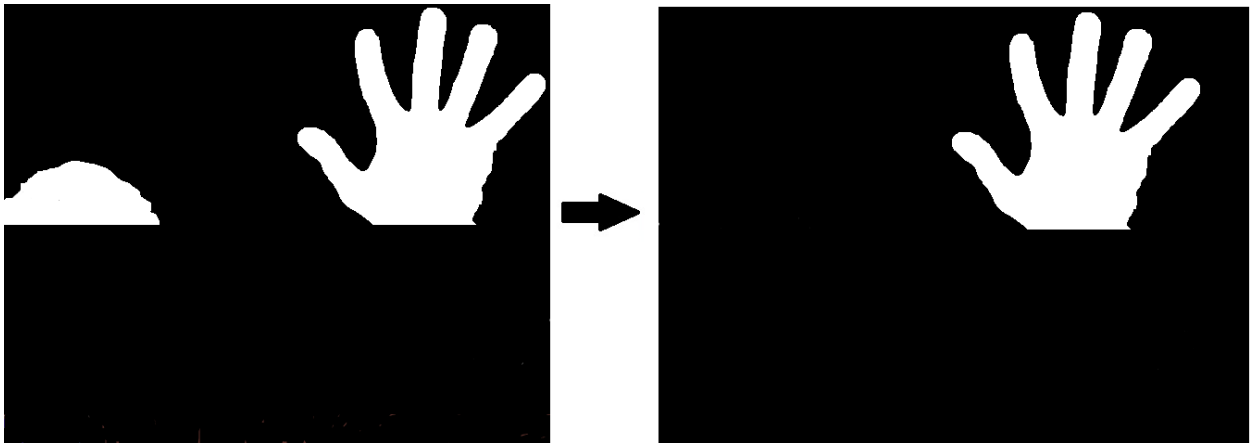


Результат удаления нижней части изображения

Выделение контура.

Задача о выделении контуров на изображении является очень распространенной в компьютерном зрении и имеет множество решений различной сложности. Но для рассматриваемой нами задачи достаточно использование выделения контуров с помощью применения морфологии, рассмотренной выше. Если применить к изображению морфологическую операцию «Сужение» и отнять от исходного изображение результирующее, то в результате останутся только контуры объектов исходного изображения.

Сделаем еще одно допущение, что контур, ограничивающий фигуру наибольшей площади – рука. Также будем считать, что рука в принципе не может иметь площадь ниже некоторого порога (автор использует порог порядка 7000 пикселей при получаемом изображении с веб-камеры размера 640x480). Исходя из этого, пользователю рекомендуется расположиться таким образом, чтобы его голова целиком не попадала в верхнюю часть изображения.



Результат фильтрации фигур по площади

Прим. Для распознавания и удаления лица пользователя с изображения можно использовать каскады Хаара. Но они имеют достаточно большую вычислительную сложность и эффективны при использовании технологии CUDA, которой автор на данный момент не располагает.

Построение модели.

Построение выпуклой оболочки.

На данном шаге нам необходимо построить выпуклую оболочку вокруг контура, который задает руку пользователя. Будем использовать алгоритм Грэхема.

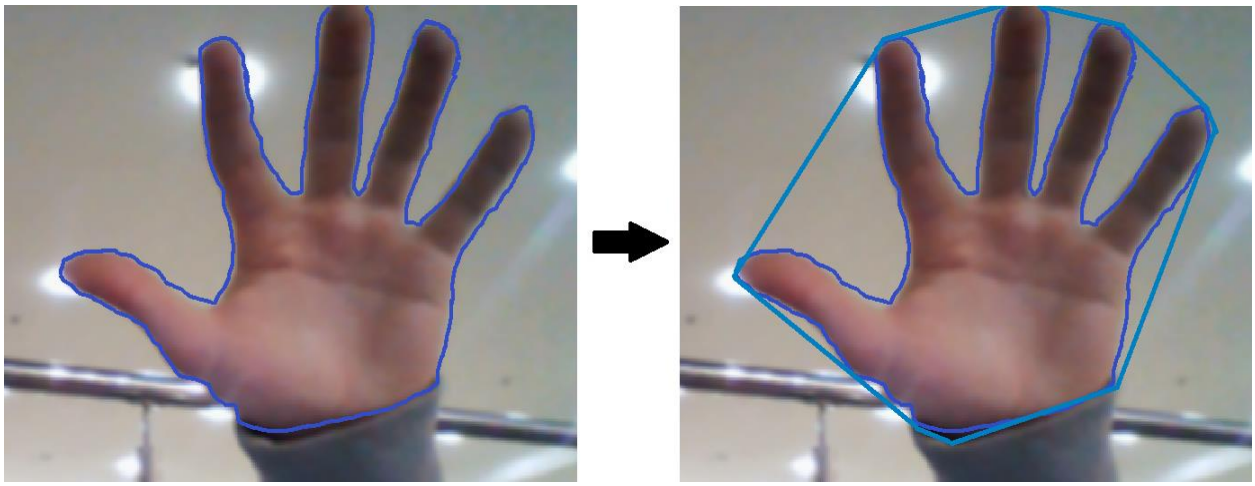
На вход алгоритму поступает множество точек Q, где |Q| >= 3 – контур, выделенный нами ранее.

Псевдокод алгоритма Грэхема выглядит следующим образом:

```
Graham(Q)
1) Пусть p_0 – точка из множества Q с минимальной координатой y или самая левая из таких точек при наличии совпадений
2) Пусть < p_1, p_2, ..., p_m > – остальные точки множества Q, отсортированные в порядке возрастания полярного угла,
     измеряемого против часовой стрелки относительно точки p_0
    (если полярные углы нескольких точек совпадают, то по расстоянию до точки p_0)
3) Push(p_0,S)
4) Push(p_1,S)
5) for i = 2 to m do
6)     while угол, образованный точками NextToTop(S),Top(S) и p_i, образуют не левый поворот
           (при движении по ломаной, образованной этими точками, мы движемся прямо или вправо)
7)       do Pop(S)
8)    Push(p_i,S)
9) return S
```

После получения выпуклой оболочки необходимо ее отфильтровать, убрав отрезки меньшие определенного коэффициента, взятого относительно размера минимального прямоугольника, ограничивающего наш контур (его легко построить по крайним точкам с каждой стороны), построить полезно не только для этого этапа (автор отсеивает отрезки меньшие 1/10 ширины такого прямоугольника).

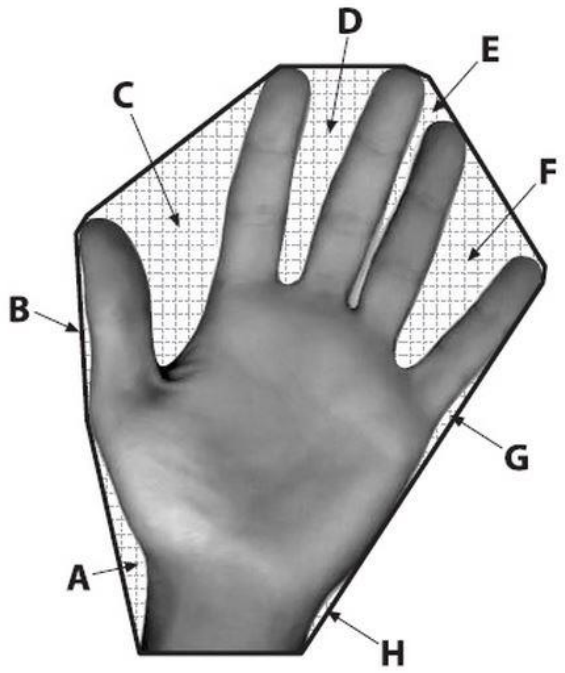
Прим. Последующая работа алгоритма ведется с бинарной маской руки, но для наглядности используются цветные иллюстрации. Пусть это не вводит читателя в заблуждение.



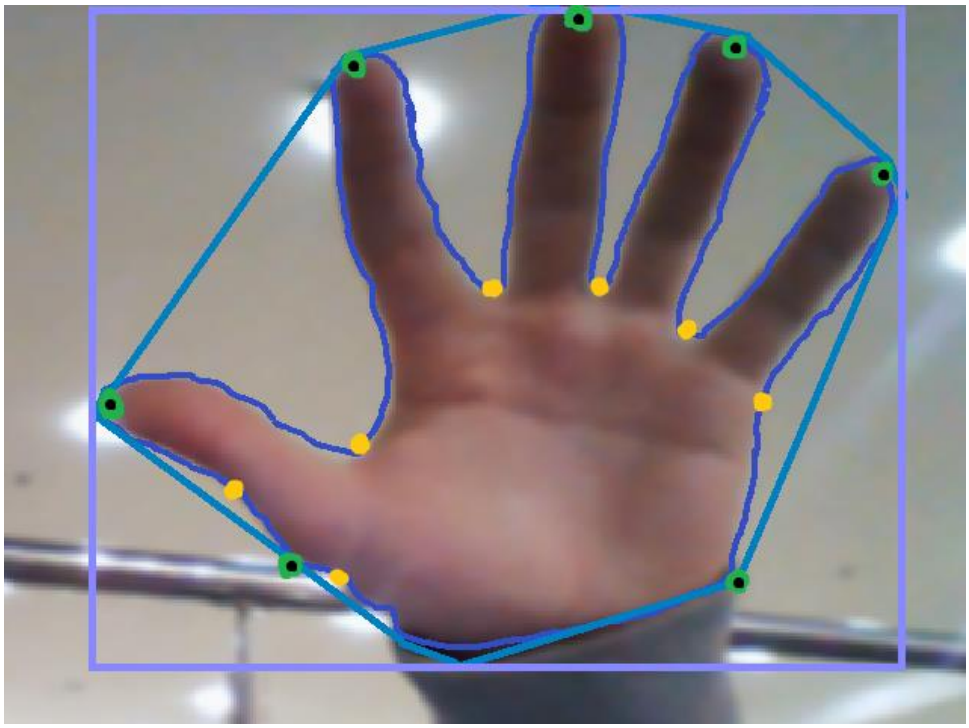
Результат построения выпуклой оболочки

Дефект выпуклости.

Дефектом выпуклости называется та площадь фигуры, которая находится между контуром, вокруг которого задается эта выпуклость и самой выпуклостью. Каждый отдельно взятый дефект выпуклости задается началом и концом отрезка, под которым он лежит, а также наиболее отдаленной от него точкой – глубиной выпуклости.



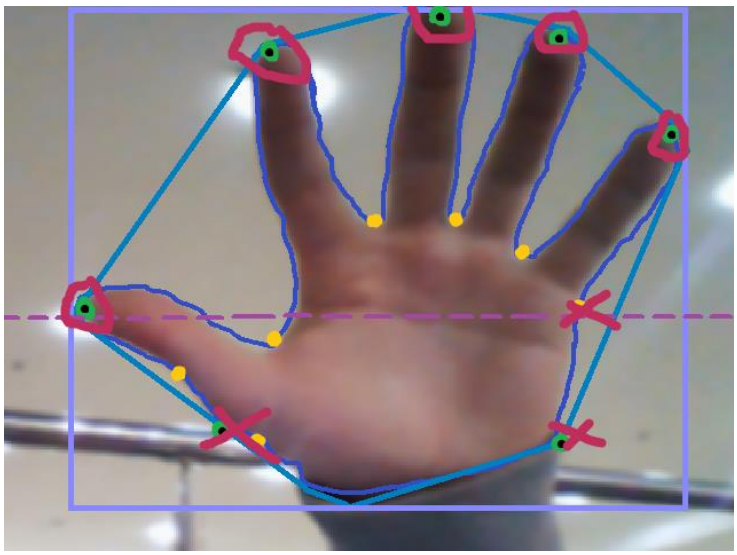
Дефекты выпуклости обозначены латинскими буквами A-H.



Концы дефектов выпуклости обозначены зелеными точками, глубины – желтыми.

Отсечение.

Теперь предлагается провести мнимую линию на некотором уровне, определяемом коэффициентом, умноженным на высоту прямоугольника. Уберем из рассмотрения все дефекты, которые полностью оказались ниже этой линии. Также отсечем те дефекты, у которых начало расположилось выше к.



Как можно заметить, оставшиеся концы дефектов выпуклости расположились на кончиках пальцев. Это не случайно. На практике подобный метод подсчета пальцев действительно работает до некоторой степени хорошо.

Анализ построенной модели.

Мы имеем построенную модель, за значениями которой теперь будем смотреть и делать необходимые выводы. Здесь стоит отметить, что здесь многие коэффициенты должны уже подстраиваться под пользователя (так же, как чувствительность мыши, например) и под его конкретное оборудование (так как многие коэффициенты имеют смысл только для конкретного значения fps, например).

Воспроизводить.

Автор предлагает в качестве команды «play» использовать жест открытой ладони. Открытая ладонь по сути своей – демонстрация 5 пальцев (без движения). Также предлагается считать, что жест должен быть продемонстрирован в течение хотя бы 3-х кадров подряд (при fps порядка 10), чтобы быть распознанным. Для уменьшения погрешности можно считать открытой ладонь из 4-х или даже 3-х пальцев. Мы можем себе это позволить, так как количество пальцев непосредственно используется в распознавании только 2-х жестов. Полезно также ограничивать область распознавания жеста, например, правым верхним углом (для правши).

Приостановить.

В качестве команды «pause» предлагается использовать демонстрация кулака – 0 пальцев. Пользователь как бы забирает воспроизведение в кулак. Здесь значение в 0 пальцев является строгим.

Следующий / предыдущий трек.

Переход к следующему треку осуществляется путем вращения кисти по часовой стрелке, в ходе которого кисть перемещается из левой части изображения в правую с достаточно высокой скоростью (автор использует порядка 40 пикселей в кадр) на протяжении «хорошего» количества кадров. Перемещение кисти отслеживается через правую границу минимального прямоугольника.

Переход к предыдущему треку осуществляется аналогичным образом в обратную сторону.

Увеличение / уменьшение громкости.

Увеличения громкости воспроизведения осуществляется путем проведения кисти от центра экрана к верхней части экрана с достаточно высокой скоростью. При выполнении данного жеста кисть должна перемещаться в вертикальном направлении с минимальным отклонением.

Уменьшение громкости осуществляется обратным действием.

Видео.

Ознакомиться с демонстрацией работы программы можно по следующей ссылке:

https://www.youtube.com/watch?v=XxgdNlThkTg&feature=youtu.be

Спасибо за внимание!