Федеральное государственное бюджетное образовательное учреждение высшего профессионального образование «Российский экономический университет имени Г. В. Плеханова» Факультет математической экономики, статистики и информатики

Курсовая работа на тему:

«Распознавание движений пальцев рук   
на основе цветовой дифференциации»

Выполнил студент

группы ДКО-131б:

Серёгин Д.А.

Проверил:

Чапкин Н.С.

Москва 2016

Оглавление

[Введение 3](#_Toc444354242)

[Постановка задачи 3](#_Toc444354243)

[HSB-цветовая модель. 4](#_Toc444354244)

[Краткое отступление 4](#_Toc444354245)

[Описание цветовой модели 4](#_Toc444354246)

[DBSCAN - плотностный алгоритм кластеризации пространственных данных с присутствием шума 6](#_Toc444354247)

[Краткое отступление 6](#_Toc444354248)

[Алгоритм работы 6](#_Toc444354249)

[Примеры работы DBSCAN-алгоритма 10](#_Toc444354250)

[Задача 11](#_Toc444354251)

[Описание задачи 11](#_Toc444354252)

[Алгоритм работы библиотеки: 11](#_Toc444354253)

[Скриншоты работы программы 13](#_Toc444354254)

[Программа только с распознаванием жестов (в правом окне выводит активный жест) 13](#_Toc444354255)

[Программа для рисования 14](#_Toc444354256)

[15](#_Toc444354257)

[Классы и описание методов 16](#_Toc444354258)

[Диаграмма классов BitMap32.dll (обработка изображения) 16](#_Toc444354259)

[Описание основных методов WebCamCapture.dll (захват изображения) 17](#_Toc444354260)

[Диаграмма классов WebCamCapture.dll (захват изображения) 19](#_Toc444354261)

[Описание основных методов WebCamCapture.dll (захват изображения) 19](#_Toc444354262)

[Заключение 21](#_Toc444354263)

[Листинг (BitMap32.dll) 22](#_Toc444354264)

[Листинг (WebCamCapture.dll) 51](#_Toc444354265)

[Листинг (Paint with WebCam) 60](#_Toc444354266)

[Список литературы 70](#_Toc444354267)

#### Введение

Человек постоянно ищет новые способы взаимодействия с компьютером, создавая все более изящные, простые устройства ввода информации в компьютер. Если компьютерная мышь была изобретена сравнительно недавно, примерно в 1970-х года 20 века, то примерная дата создания клавиатуры уходит в 19 век. Но сейчас на дворе 21 век, и сейчас мы имеем гораздо больший ассортимент выбора способа взаимодействия с компьютером: сканер, джойстик, тачпад, веб-камера, голос, даже считывание активности головного мозга.

Многие из этих способов считаются уже довольно обыденными, некоторые до сих пор остаются самыми удобными, например компьютерная мышь или тачпад.

Но что, если попробовать разнообразить общение с ПК с помощью жестов. То есть управлять компьютером с помощью движения пальцев рук.

Эта идея получила широкое использование и вылилась в такую сферу, как «Компьютерное зрение». С помощью него, можно отслеживать движения пальцев, кистей, рук, движение любых конечностей, распознавание эмоций на лице человека и так далее.

Возьмем наиболее простую в реализации (за счет масштабов и простоты геометрических линий) разновидность - движение пальцев, кистей. Но чтобы еще немного упростить задачу, добавим обязательным условием разделение пальцев по цветам (перчатки, цветные бумажки или же фломастерами нарисовать на пальцах различные цвета).

Итак, попробуем написать свою dll-библиотеку для распознавания этих самых пальцев.

##### Постановка задачи

Требуется написать dll-библиотеку, которая будет распознавать и анализировать движение пальцев рук на основе цветовой дифференциации, обрабатывая изображение, полученное с веб-камеры. Затем, на основе данной библиотеки создать программу, которая показывает практическое применение, например обычный Paint.

#### HSB-цветовая модель.

##### Краткое отступление

Среди большого разнообразия цветовых моделей (RGB, LAB, XYZ, CMYK, HSB (HSV), HSL) была выбрана именно HSB (HSV) - модель, так как она считается наиболее приближенной к тому, как цвет воспринимает человеческий глаз.

А так же за счет простоты обработки цвета (по сравнению с RGB-моделью, где требуется анализировать три показателя цвета, здесь на цветовой тон отводится только один показатель, что существенно упрощает реализацию цветового фильтра).

##### Описание цветовой модели

HSB (Hue - оттенок, Saturation - насыщенность, Brightness - яркость) цветовая модель определяет цветовое пространство, как соответствии трех составляющих:

Оттенок: тип цвета (например красный, желтый или зелёный).

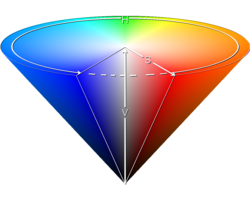
Варьируется от 0 до 360° (каждое значение соответствует одному цвету : 0 - красный, 45 - оттенок оранжевого и 55 - оттенок желтого).

Насыщенность: интенсивность цвета.

Варьируется от 0 до 100% (0 означает отсутствие цвета, то есть оттенок серого между черным и белым; 100 означает интенсивный цвет).

Так же иногда называют «purity» - «чистота» по аналогии с колориметрическим показателем величины возбуждения.

Яркость (или значение) : яркость цвета.

Варьируется от 0 до 100% (0 всегда черный; в зависимости от насыщенности, 100 может быть белым или более или менее насыщенным цветом).

Её общее графическое представление представлено так:

HSB-модель также называют HSV-модель (Hue - оттенок, Saturation - насыщенность, Value - значение). HSV-модель была создана в 1978 Альви Рэй Смитом. Она представляет собой нелинейное преобразование всем известной RGB-модели. Другими словами, цвет не может быть задан как простая операция (сложение/вычитание) исходного цвета, а может как математическая трансформация.

#### DBSCAN - плотностный алгоритм кластеризации пространственных данных с присутствием шума

##### Краткое отступление

Если в выборе цветовой модели у нас в распоряжении было всего несколько вариантов, то здесь выбор в разы больше.

После небольшого сравнительного анализа, был выбран именно DBSCAN - алгоритм, так как он наиболее приемлемо для текущей задачи справляется с «выбросами» - различными шумами (точки, случайно прошедшие отбор цветовым фильтром), а так же за счет приемлемой скорости работы и относительно простой реализации.

##### Алгоритм работы

Прежде всего нужно задать некое число Эпсилон, которое является неким радиусом у одной точки и минимальное количество соседних точек, которые лежат в окружности радиуса Эпсилон от «исходной» точки.

Затем требуется задать функцию, для определения расстояния между точками (метрика), саму функцию DBSCAN и функцию расширения кластера (далее приведен псевдокод):

*DBSCAN(D, eps, MinPts)*

*C = 0*

*для каждой непосещённой точки P в наборе D*

*пометить P как посещенную*

*N = получитьСоседей (P, eps)*

*if sizeof(N) < MinPts*

*пометить P как шум*

*else*

*C = следующий кластер*

*расширитьКластер(P, N, C, eps, MinPts)*

*расширитьКластер(P, N, C, eps, MinPts)*

*добавить P к кластеру C*

*для каждой точки P' в N*

*если P' не посещена*

*пометить P' как посещённую*

*N' = получитьСоседей(P', eps)*

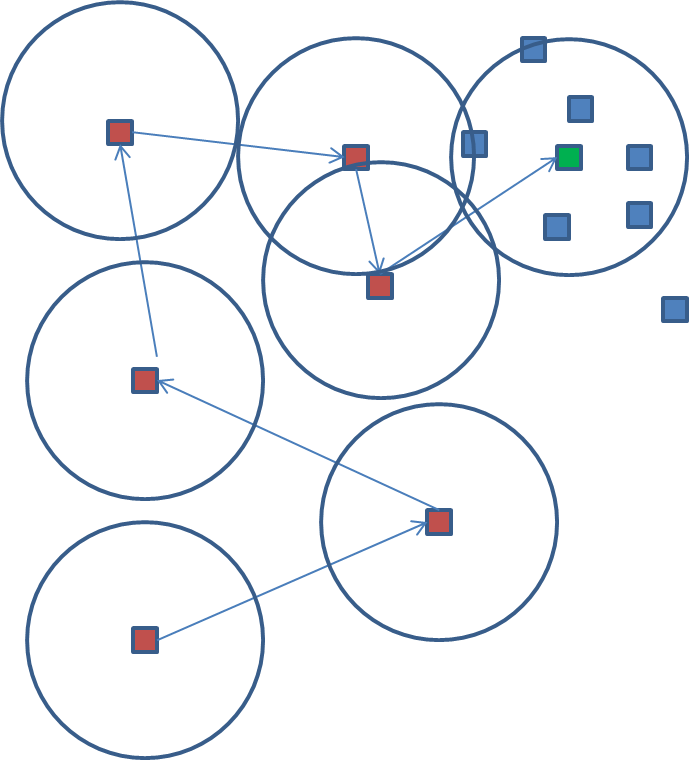
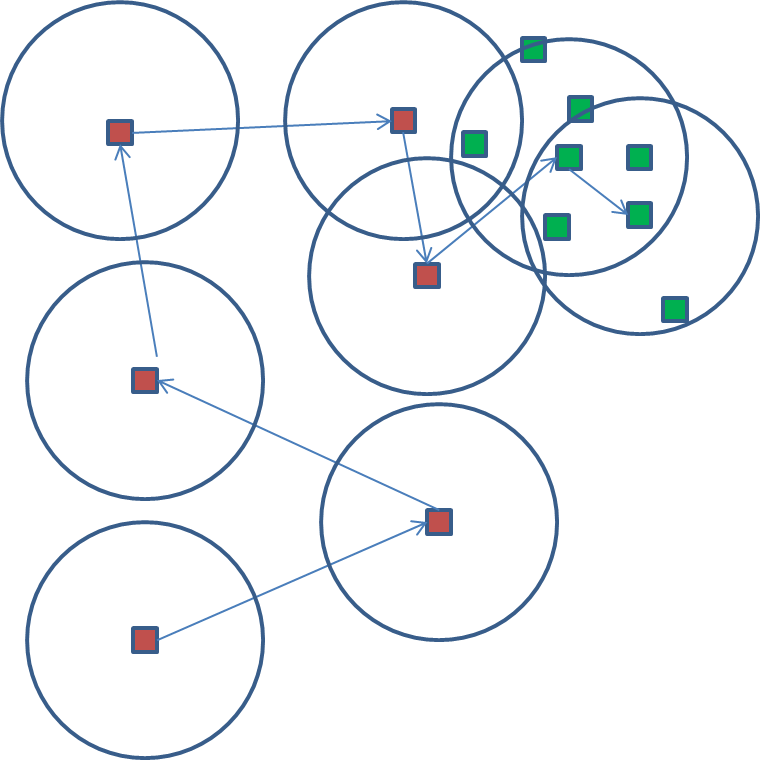
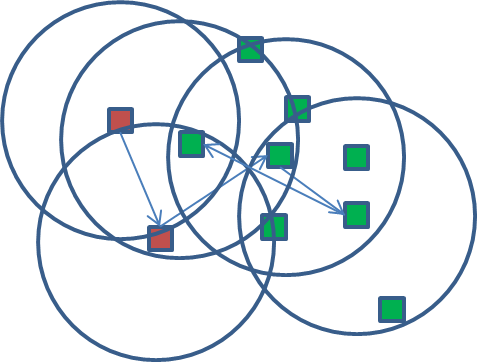
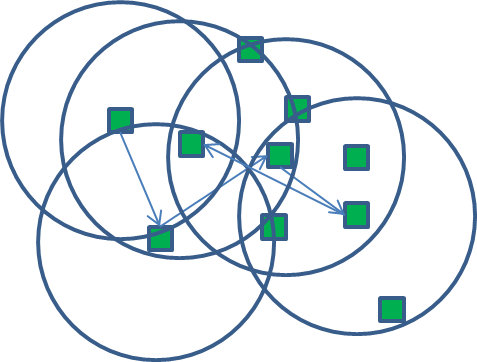
*if sizeof(N') >= MinPts*

*N = к N присоединить N'*

*если P' все еще не принадлежит никакому кластеру*

*добавить P' к кластеру C*

Или же более визуально:

* У нас имеется список точек.
* Первое: выберем первую точку случайно  
    
  Проверить окружность точки (Эпсилон - радиус круга) на наличие в ней 4 соседних точек. Если проверка пройдена, инициировать кластер (зелёный), иначе пометить как шум(красный) (функция DBSCAN для каждой не посещённой точки). Стрелки показывают все точки, которые мы посетили.
* Второе: Расширить кластер  
  Как только нашли кластер, пометили все точки зеленым и проверяем соседние точки, как подходящие или нет для этого кластера
* Заметьте: точка, имеющая ранее статус «Шум» может стать зеленой (быть присоединена к кластеру)
* Две красные точки на самом деле в кластере
* После того, как прошли все точки, останавливаемся.

##### Примеры работы DBSCAN-алгоритма

Рисунок 3 ДО

Рисунок 1 После\*

Рисунок 2 До



Рисунок 4 После\*

Под «После\*» понимается не только класстеризация, но и выделение среди класстеров одного, с наибольшим количеством точек (делается допущение, что шумы не могут быть расположены так же плотно, как цветовые сегменты пальца)

#### Задача

##### Описание задачи

По условию задачи, у нас есть только веб-камера, с помощью которой мы должны осуществлять функции ввода информации в компьютер. Другими словами, мы создаем библиотеку, которая забирает изображение с веб-камеры, анализирует кадр, находит расположение пальцев и в каком-либо удобном виде выдает координаты этих пальцев.

Затем, на основе координат пальцев происходит программное взаимодействие с курсором, то есть пальцы представляют из себя многофункциональную компьютерную мышь (больше сочетаний-«кнопок», перемещение в трех-мерном пространстве).

##### Алгоритм работы библиотеки:

1. Библиотечная функция, с помощью вызова нативных WinApi-методов копирует изображение в буфер обмена и затем забирает его оттуда, для дальнейшей обработки
2. Происходит первичная обработка полученного кадра
   1. Попиксельный проход по всему изображению с преобразование пикселей в HSB-Цветовую модель и сохранением пикселей, который подходят заранее подготовленному цветовому фильтру (HSB-значения) в отдельный список.
   2. Каждый список пикселей для одного пальца сохраняется в свою очередь в список всех пальцев.
3. Затем найденные первичные сектора подвергаются дальнейшей обработке: кластерному анализу.
   1. В каждом списке для отдельного файла выделяем один сектор при помощи плотностного алгоритма кластеризации пространственных данных с присутствием шума (DBSCAN).
   2. В полученном секторе высчитываем среднее значение цвета и в соответствии с цветовым фильтром (см. п. 2.1) сохраняются на определенную позицию. Каждая позиция зафиксирована за одним пальцев в заранее созданном массиве.
4. На исходном изображении, полученном с камеры, рисуются прямоугольные сектора, созданные на основе каждого цветного сектора соответствующего пальца.
5. С помощью библиотечного метода возвращается массив пальцев, который затем передается в обработчик жестов
   1. Для каждого цветового сектора вновь высчитывается прямоугольный сектор и его центр.
   2. Если расстояние между центрами прямоугольника большого пальца и какого-либо другого прямоугольника меньше 50, тогда возвращается соответствующий номер активного жеста.
6. «Наверху» происходит последний этап обработки - анализирование полученных жестов
   1. Проверяется номер текущего жеста, и если он не совпадает с предыдущим, значит пользователь произвел некое действие - сложил пальцы в новом жесте.
   2. Обрабатываем этот жест, имитируя например нажатие клавиш мыши или вешая на них некие обработчики, например смену цвета или ширины кисти.ц

#### Скриншоты работы программы

##### Программа только с распознаванием жестов (в правом окне выводит активный жест)

##### Программа для рисования

# 

#### Классы и описание методов

##### Диаграмма классов BitMap32.dll (обработка изображения)

##### Описание основных методов WebCamCapture.dll (захват изображения)

|  |  |
| --- | --- |
| Метод | Описание |
| **public void HSBFindColoredPlots()** | Алгоритм первичного поиска цветных секторов по всему изображению |
| **private void m\_Optimaze\_Plots()** | Кластеризация полученных секторов и выделение одного с наибольшим количеством точек |
| **void LockBitmap()** | Блокирует пиксели на изображении для ускорения производительности |
| **void LockBitmap()** | Разблокирует пиксели: копирует данные обратно в Bitmap и освобождает ресурсы |
| **public Color GetPixel(BitmapData m\_BitmapData, int x, int y)** | Возвращает цветовые характеристики (R, G, B) и степень прозрачности (A) |
| **public void DrawColoredPlots(string nameFile)** | Рисует цветовые сектора и сохраняет по указанному адресу. |
| **public static Bitmap LoadBitmap(string fileName)** | Загружает изображение как Bitmap из указанного файла |
| **public Bitmap ImageWithColoredPlots()** | Возвращает изображение как Bitmap с нарисованными цветными секторами |
| **public List<DatasetItem>[] GetFingers()** | Возвращает массив списков пикселей для всех 5 пальцев |
| **GestureNumberHelper public static string ToString(int gestureNumber)** | Преобразует номер жеста в текстовое наименование |
| **public int CheckGesture(List<DatasetItem>[] fingers)** | Возвращает номер текущего жеста |
| **int GetDistance(Point x, Point y)** | Возвращает расстояние между двумя точками (для готовых прямоугольников) |
| **public Point Center(Rectangle rect)** | Возвращает центр прямоугольного сектора |
| **public void ComputeClusterDbscan(DatasetItem[] allPoints, double epsilon, int minPts, out HashSet<DatasetItem[]> clusters)** | Алгортим кластеризации DBSCAN |
| **private void ExpandCluster(DbscanPoint[] allPoints, DbscanPoint p, DbscanPoint[] neighborPts, int c, double epsilon, int minPts)** | Метод расширения кластера для алгоритма DBSCAN |
| **private void RegionQuery(DbscanPoint[] allPoints, DatasetItem p, double epsilon, out DbscanPoint[] neighborPts)** | Поиск точек, входящих в окружность радиуса epsilon |

### 

##### Диаграмма классов WebCamCapture.dll (захват изображения)

##### Описание основных методов WebCamCapture.dll (захват изображения)

|  |  |
| --- | --- |
| Метод | Описание |
| public void Start(ulong FrameNum) | Стартует захват видеоряда с веб-камеры |
| public void Stop() | Останавливает захват видеоряда с веб-камеры |
| private void timer1\_Tick(object sender, System.EventArgs e) | Захватывает следующий кадр из видеоряда |

#### Заключение

В ходе работы над курсовым проектом, была создана библиотека для обработки изображения с выделением цветовых секторов и выдачи их через публичные методы, а так же была изучена небольшая теоретическая часть (например такие интересные темы, как кластерный анализ и компьютерное зрение).

Данное ПО можно многократно оптимизировать и улучшать, так как здесь приведена лишь грубая, но рабочая версия.

Подводя итог, хотелось бы отметить, что данное направление (компьютерное зрение) - является одной из наиболее интересных областей дальнейшего развития взаимодействия компьютера и человека, которая все больше стирает грани между реальностью и виртуальностью.

#### Листинг (BitMap32.dll)

using System;

using System.Drawing;

using System.Drawing.Imaging;

using System.Runtime.InteropServices;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.IO;

**public class BitMap32**

{

/// <summary>

/// Изображение для обработки представленное в виде массива байтов.

/// </summary>

byte[] m\_imageBytes;

/// <summary>

/// Оригинальное изображение.

/// </summary>

Bitmap m\_imgOriginal;

/// <summary>

/// Изображение, куда будем сохранять цветные сектора.

/// </summary>

Bitmap m\_colors;

/// <summary>

/// Копия оригинального изображения.

/// </summary>

BitmapData m\_copyOriginalImg;

/// <summary>

/// Заблокировано ли изображение или нет.

/// </summary>

bool m\_IsLocked;

/// <summary>

/// Списки, куда будем сохранять подходящие точки.

/// </summary>

List<FingerPoint> Red;

List<FingerPoint> Green;

List<FingerPoint> Purple;

List<FingerPoint> Yellow;

List<FingerPoint> Blue;

List<List<FingerPoint>> Fingers;

/// <summary>

/// Прямоугольники для пальцев.

/// </summary>

List<DatasetItem>[] FingersForGesture = new List<DatasetItem>[5];

/// <summary>

/// Создает экземпляр класса <see cref="WorkWithJpg.BitMap32"/>.

/// </summary>

/// <param name="path">Путь к файлу для открытия.</param>

public BitMap32(string path)

{

m\_imgOriginal = LoadBitmap(path);

m\_BitMap32();

}

/// <summary>

/// Создает экземпляр класса <see cref="WorkWithJpg.BitMap32"/>.

/// </summary>

/// <param name="image">Изображение, с которого взять информацию.</param>

public BitMap32(Image image)

{

m\_imgOriginal = new Bitmap(image);

m\_BitMap32();

}

/// <summary>

/// Приватный конструктор на основе которого строятся публичные.

/// </summary>

private void m\_BitMap32() {

m\_colors = new Bitmap(m\_imgOriginal.Width, m\_imgOriginal.Height);

Red = new List<FingerPoint>();

Green = new List<FingerPoint>();

Blue = new List<FingerPoint>();

Yellow = new List<FingerPoint>();

Purple = new List<FingerPoint>();

Fingers = new List<List<FingerPoint>>();

Fingers.Add(Red);

Fingers.Add(Green);

Fingers.Add(Blue);

Fingers.Add(Yellow);

Fingers.Add(Purple);

for (int i = 0; i < FingersForGesture.Length; i++) {

FingersForGesture[i] = new List<DatasetItem>();

}

m\_IsLocked = false;

}

/// <summary>

/// Алгоритм первичного поиска цветных секторов по всему изображению

/// </summary>

public void HSBFindColoredPlots()

{

// Блокируем пиксели для более быстрого доступа

LockBitmap();

// Пробегаемся по всему изображению в двойном цикле

for (int i = 0; i < m\_copyOriginalImg.Width; i += 5)

{

for (int j = 0; j < m\_copyOriginalImg.Height; j += 5)

{

Color pixel = GetPixel(m\_copyOriginalImg, i, j);

HSB hsbpixel = Conversion.RGBtoHSB(pixel.R, pixel.G, pixel.B);

if (hsbpixel.Brightness >= 0.3 && hsbpixel.Saturation >= 0.2)

{

// Case Yellow

if (hsbpixel.Saturation >= 0.3 && ((hsbpixel.Hue >= 40.0) && (hsbpixel.Hue <= 70.0)))

{

Yellow.Add(new FingerPoint(i, j, pixel, hsbpixel));

m\_colors.SetPixel(i, j, pixel);

continue;

}

// Case Green

if (hsbpixel.Hue >= 70.0 && hsbpixel.Hue <= 160.0)

{

Green.Add(new FingerPoint(i, j, pixel, hsbpixel));

m\_colors.SetPixel(i, j, pixel);

continue;

}

// Case Red

if (hsbpixel.Saturation >= 0.5 && (hsbpixel.Hue >= 355.0 || hsbpixel.Hue <= 18.0))

{

Red.Add(new FingerPoint(i, j, pixel, hsbpixel));

m\_colors.SetPixel(i, j, pixel);

continue;

}

// Case Blue

if (hsbpixel.Saturation >= 0.5 && hsbpixel.Hue >= 210.0 && hsbpixel.Hue <= 240.0)

{

Blue.Add(new FingerPoint(i, j, pixel, hsbpixel));

m\_colors.SetPixel(i, j, pixel);

continue;

}

// Case Purple

if (hsbpixel.Hue >= 300.0 && hsbpixel.Hue <= 355.0)

{

Purple.Add(new FingerPoint(i, j, pixel, hsbpixel));

m\_colors.SetPixel(i, j, pixel);

continue;

}

}

}

}

// Разблокируем изображением.

UnlockBitmap();

// Класстеризуем каждый из найденных секторов

m\_Optimaze\_Plots();

}

private void m\_Optimaze\_Plots() {

// Создаем новый список пальцев (цветных секторов)

List<DatasetItem[]> NewFingers = new List<DatasetItem[]>();

// Пробегаемся по первичным секторам

foreach (var oldFinger in Fingers) {

// Сюда выделенные области

HashSet<DatasetItem[]> clusters = new HashSet<DatasetItem[]>();

// Переводим наш список Points в список DatasetItem

List<DatasetItem> Cluster = new List<DatasetItem>();

// Добавляем в список всех точек наши точки одного цвета из первичного сектора

foreach (var item in oldFinger) {

Cluster.Add(new DatasetItem(item.Point.X, item.Point.Y, item.Color, item.HSBColor));

}

// Создаем алгоритм подсчитывания расстояния (метрики)

var dbs = new DbscanAlgorithm((x, y) => Math.Sqrt(((x.X - y.X) \* (x.X - y.X)) + ((x.Y - y.Y) \* (x.Y - y.Y))));

// Выделяем из первичного сектора кластеры

dbs.ComputeClusterDbscan(allPoints: Cluster.ToArray(), epsilon: 7.5, minPts: 8, clusters: out clusters);

if (clusters.Count != 0)

{

// Ищем кластер с наибольшем количеством точек

var maxPoints = clusters.Max(item => item.Length);

// И запоминаем его

var finger = clusters.First(item => item.Length == maxPoints);

NewFingers.Add(finger);

// Смотрим, какой цвет у пальца

var fingerSaturation = finger.Average(item => item.HSBColor.Hue);

if ((fingerSaturation >= 40.0) && (fingerSaturation <= 70.0)) // Case Yellow

FingersForGesture[(int) FingersColor.YELLOW] = finger.ToList();

else if (fingerSaturation >= 70.0 && fingerSaturation <= 160.0) // Case Green

FingersForGesture[(int) FingersColor.GREEN] = finger.ToList();

else if (fingerSaturation >= 355.0 || fingerSaturation <= 18.0) // Case Red

FingersForGesture[(int) FingersColor.RED] = finger.ToList();

else if (fingerSaturation >= 210.0 && fingerSaturation <= 240.0) // Case Blue

FingersForGesture[(int) FingersColor.BLUE] = finger.ToList();

else if (fingerSaturation >= 300.0 && fingerSaturation <= 355.0) // Case Purple

FingersForGesture[(int) FingersColor.PURPLE] = finger.ToList();

}

}

// Рисуем квадраты на исходном изображении

using (var graphics = Graphics.FromImage(m\_imgOriginal))

{

foreach (var item in NewFingers) {

// Считываем цвет первого пикселя из массива

LockBitmap();

Color pixel = GetPixel(m\_copyOriginalImg, (int) item[0].X, (int) item[0].Y);

UnlockBitmap();

// Задаем цвет рамки, равный цвету первого пикселя

Pen Pen = new Pen(Color.FromArgb(pixel.R, pixel.G, pixel.B), 2);

if (item.Length != 0) {

// Draw rect

graphics.DrawRectangle(Pen,

(float) item.Min(a => a.X),

(float) item.Min(a => a.Y),

(float) item.Max(a => a.X) - (float) item.Min(a => a.X),

(float) item.Max(a => a.Y) - (float) item.Min(a => a.Y));

}

}

//m\_imgOriginal.Save("clusters/test5\_4/bla.jpg");

}

}

/// <summary>

/// Блокирует пиксели на изображении.

/// </summary>

void LockBitmap()

{

// Если уже заблокированы, то ничего не делать.

if (m\_IsLocked)

return;

// Создаем прямоугольник для блокирования

Rectangle bounds = new Rectangle(0, 0,

m\_imgOriginal.Width, m\_imgOriginal.Height);

// Блокирует данные на оригинальном изображении (накрывая прямоугольником)

m\_copyOriginalImg = m\_imgOriginal.LockBits(bounds,

ImageLockMode.ReadWrite,

PixelFormat.Format32bppArgb);

// Распределить память для данных.

int total\_size = m\_copyOriginalImg.Stride \* m\_copyOriginalImg.Height;

m\_imageBytes = new byte[total\_size];

// Копировать данные в массив ImageBytes.

Marshal.Copy(m\_copyOriginalImg.Scan0, m\_imageBytes, 0, total\_size);

m\_IsLocked = true;

}

/// <summary>

/// Разблокирует пиксели: копирует данные обратно в Bitmap и освобождает ресурсы.

/// </summary>

void UnlockBitmap()

{

// Уже разблокировано, ничего не делать.

if (!m\_IsLocked)

return;

// Копировать данные обратно в bitmap.

int total\_size = m\_copyOriginalImg.Stride \* m\_copyOriginalImg.Height;

Marshal.Copy(m\_imageBytes, 0, m\_copyOriginalImg.Scan0, total\_size);

// Разблокировать изображение.

m\_imgOriginal.UnlockBits(m\_copyOriginalImg);

// Освободить ресурсы.

m\_imageBytes = null;

m\_copyOriginalImg = null;

// Теперь разблокировано.

m\_IsLocked = false;

}

/// <summary>

/// Возвращает цветовые характеристики (R, G, B) и степень прозрачности (A)

/// из указанного изображения по указанным координатам.

/// </summary>

/// <param name="m\_BitmapData">Изображение, откуда брать пиксель.</param>

/// <param name="x">Координата X пикселя</param>

/// <param name="y">Координата Y пикселя.</param>

/// <param name="red">Выходной байт для красного цвета</param>

/// <param name="green">Выходной байт для зеленого цвета.</param>

/// <param name="blue">Выходной байт для синего цвета.</param>

/// <param name="alpha">Выходной байт для степени прозрачности.</param>

public Color GetPixel(BitmapData m\_BitmapData, int x, int y)

{

int i = y \* m\_BitmapData.Stride + x \* 4;

int blue = (int)m\_imageBytes[i++];

int green = (int)m\_imageBytes[i++];

int red = (int)m\_imageBytes[i++];

int alpha = (int)m\_imageBytes[i];

Color pixel = Color.FromArgb(alpha, red, green, blue);

return pixel;

}

/// <summary>

/// Рисуем цветовые сектора и сохраняем по указанному адресу.

/// </summary>

/// <param name="nameFile>Имя сохранённого файла.</param>

public void DrawColoredPlots(string nameFile)

{

// Рисуем квадраты на дополнительном изображении

using (var graphics = Graphics.FromImage(m\_colors))

{

Pen Pen = new Pen(Color.Black, 1);

foreach (var item in Fingers) {

if (item.Count != 0) {

// Draw rect

graphics.DrawRectangle(Pen,

item.Min(a => a.Point.X),

item.Min(a => a.Point.Y),

item.Max(a => a.Point.X) - item.Min(a => a.Point.X),

item.Max(a => a.Point.Y) - item.Min(a => a.Point.Y));

}

}

m\_colors.Save(nameFile + "colors\_painted.png");

// Console.WriteLine("Results saved!");

}

// Рисуем квадраты на исходном изображении

using (var graphics = Graphics.FromImage(m\_imgOriginal))

{

Pen Pen = new Pen(Color.Black, 1);

foreach (var item in Fingers) {

if (item.Count != 0) {

// Draw rect

graphics.DrawRectangle(Pen,

item.Min(a => a.Point.X),

item.Min(a => a.Point.Y),

item.Max(a => a.Point.X) - item.Min(a => a.Point.X),

item.Max(a => a.Point.Y) - item.Min(a => a.Point.Y));

}

}

m\_imgOriginal.Save(nameFile + "original\_colors\_painted.jpg");

// Console.WriteLine("Results saved!");

}

}

/// <summary>

/// Загружает изображение как Bitmap из указанного файла.

/// </summary>

/// <returns>The bitmap.</returns>

/// <param name="fileName">File name.</param>

public static Bitmap LoadBitmap(string fileName)

{

using (FileStream fs = new FileStream(fileName, FileMode.Open, FileAccess.Read, FileShare.Read))

return new Bitmap(fs);

}

public Bitmap ImageWithColoredPlots()

{

return m\_imgOriginal;

}

public List<DatasetItem>[] GetFingers()

{

return FingersForGesture;

}

}

using System;

//Dbscan clustering identifiers

public enum ClusterIds

{

UNCLASSIFIED = 0,

NOISE = -1

}

using System;

public enum FingersColor

{

YELLOW = FingersName.THUMB,

BLUE = FingersName.INDEXFINGER,

RED = FingersName.MIDDLEFINGER,

GREEN = FingersName.RINGFINGER,

PURPLE = FingersName.PINKY

}

using System;

public enum FingersName

{

THUMB = 0,

INDEXFINGER = 1,

MIDDLEFINGER = 2,

RINGFINGER = 3,

PINKY = 4

}

using System;

public enum GestureNumber

{

NOTHING\_HAS\_DETECTED = -1,

THUMB\_INDEX\_FINGER = 1,

THUMB\_MIDDLE\_FINGER = 2,

THUMB\_RING\_FINGER = 3,

THUMB\_PINKY = 4

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

abstract class Conversion

{

/// <summary>

/// Converts RGB to HSB.

/// </summary>

public static HSB RGBtoHSB(int red, int green, int blue)

{

// normalize red, green and blue values

double r = ((double)red / 255.0);

double g = ((double)green / 255.0);

double b = ((double)blue / 255.0);

// conversion start

double max = Math.Max(r, Math.Max(g, b));

double min = Math.Min(r, Math.Min(g, b));

double h = 0.0;

if (max == r && g >= b)

{

h = 60 \* (g - b) / (max - min);

}

else if (max == r && g < b)

{

h = 60 \* (g - b) / (max - min) + 360;

}

else if (max == g)

{

h = 60 \* (b - r) / (max - min) + 120;

}

else if (max == b)

{

h = 60 \* (r - g) / (max - min) + 240;

}

double s = (max == 0) ? 0.0 : (1.0 - (min / max));

return new HSB(h, s, (double)max);

}

/// <summary>

/// Converts HSB to RGB.

/// </summary>

public static RGB HSBtoRGB(double h, double s, double v)

{

double r = 0;

double g = 0;

double b = 0;

if (s == 0)

{

r = g = b = v;

}

else

{

// the color wheel consists of 6 sectors. Figure out which sector

// you're in.

double sectorPos = h / 60.0;

int sectorNumber = (int)(Math.Floor(sectorPos));

// get the fractional part of the sector

double fractionalSector = sectorPos - sectorNumber;

// calculate values for the three axes of the color.

double p = v \* (1.0 - s);

double q = v \* (1.0 - (s \* fractionalSector));

double t = v \* (1.0 - (s \* (1 - fractionalSector)));

// assign the fractional colors to r, g, and b based on the sector

// the angle is in.

switch (sectorNumber)

{

case 0:

r = v;

g = t;

b = p;

break;

case 1:

r = q;

g = v;

b = p;

break;

case 2:

r = p;

g = v;

b = t;

break;

case 3:

r = p;

g = q;

b = v;

break;

case 4:

r = t;

g = p;

b = v;

break;

case 5:

r = v;

g = p;

b = q;

break;

}

}

return new RGB(

Convert.ToInt32(Double.Parse(String.Format("{0:0.00}", r \* 255.0))),

Convert.ToInt32(Double.Parse(String.Format("{0:0.00}", g \* 255.0))),

Convert.ToInt32(Double.Parse(String.Format("{0:0.00}", b \* 255.0)))

);

}

}

using System;

using System.Drawing;

//Each items in your dataset, preferences, features, points, etc.

public class DatasetItem

{

public double X;

public double Y;

public Color Color;

public HSB HSBColor;

public DatasetItem(double x, double y, Color color, HSB hsbColor)

{

X = x;

Y = y;

Color = color;

HSBColor = hsbColor;

}

}

using System;

using System.Linq;

using System.Collections.Generic;

public class DbscanAlgorithm

{

private readonly Func<DatasetItem, DatasetItem, double> \_metricFunc;

public DbscanAlgorithm(Func<DatasetItem, DatasetItem, double> metricFunc)

{

\_metricFunc = metricFunc;

}

public void ComputeClusterDbscan(DatasetItem[] allPoints, double epsilon, int minPts, out HashSet<DatasetItem[]> clusters)

{

clusters = null;

var allPointsDbscan = allPoints.Select(x => new DbscanPoint(x)).ToArray();

var C = 0;

for (int i = 0; i < allPointsDbscan.Length; i++)

{

var p = allPointsDbscan[i];

if (p.IsVisited)

continue;

p.IsVisited = true;

DbscanPoint[] neighborPts = null;

RegionQuery(allPointsDbscan, p.ClusterPoint, epsilon, out neighborPts);

if (neighborPts.Length < minPts)

p.ClusterId = (int)ClusterIds.NOISE;

else

{

C++;

ExpandCluster(allPointsDbscan, p, neighborPts, C, epsilon, minPts);

}

}

clusters = new HashSet<DatasetItem[]>(

allPointsDbscan

.Where(x => x.ClusterId > 0)

.GroupBy(x => x.ClusterId)

.Select(x => x.Select(y => y.ClusterPoint).ToArray())

);

}

private void ExpandCluster(DbscanPoint[] allPoints, DbscanPoint p, DbscanPoint[] neighborPts, int c, double epsilon, int minPts)

{

p.ClusterId = c;

for (int i = 0; i < neighborPts.Length; i++)

{

var pn = neighborPts[i];

if (!pn.IsVisited)

{

pn.IsVisited = true;

DbscanPoint[] neighborPts2 = null;

RegionQuery(allPoints, pn.ClusterPoint, epsilon, out neighborPts2);

if (neighborPts2.Length >= minPts)

{

neighborPts = neighborPts.Union(neighborPts2).ToArray();

}

}

if (pn.ClusterId == (int)ClusterIds.UNCLASSIFIED)

pn.ClusterId = c;

}

}

private void RegionQuery(DbscanPoint[] allPoints, DatasetItem p, double epsilon, out DbscanPoint[] neighborPts)

{

neighborPts = allPoints.Where(x => \_metricFunc(p, x.ClusterPoint) <= epsilon).ToArray();

}

}

using System;

//Point container for Dbscan clustering

public class DbscanPoint

{

public bool IsVisited;

public DatasetItem ClusterPoint;

public int ClusterId;

public DbscanPoint(DatasetItem x)

{

ClusterPoint = x;

IsVisited = false;

ClusterId = (int)ClusterIds.UNCLASSIFIED;

}

}

using System;

using System.Drawing;

public struct FingerPoint

{

public Point Point {

get;

set;

}

public Color Color {

get;

set;

}

public HSB HSBColor {

get;

set;

}

public FingerPoint(int x, int y, Color color, HSB hsbColor)

{

Point = new Point(x, y);

Color = color;

HSBColor = hsbColor;

}

}

using System;

using System.Drawing;

using System.Collections.Generic;

using System.Linq;

public struct FingerRectangle

{

public FingerRectangle(Point leftTop,

Point rightTop,

Point leftBottom,

Point rightBottom)

{

LeftTop = leftTop;

RightTop = rightTop;

LeftBottom = leftBottom;

RightBottom = rightBottom;

}

public Point LeftTop

{

get;

set;

}

public Point RightTop

{

get;

set;

}

public Point LeftBottom

{

get;

set;

}

public Point RightBottom

{

get;

set;

}

public Point Center

{

get {

return new Point(((GetRightBottomX() + GetLeftTopX())/2),

((GetRightBottomY() + GetLeftTopY())/2));

}

}

public void Check(Point test)

{

// Test Left Top

if (test.Y <= LeftTop.Y && test.X <= LeftTop.X)

{

LeftTop = test;

}

// Test RightTop

else if (test.X >= RightTop.X && test.Y <= RightTop.Y)

{

RightTop = test;

}

// Test LeftBottom

else if (test.X <= LeftBottom.X && test.Y >= LeftBottom.Y)

{

LeftBottom = test;

}

// Test RightBottom

else if (test.X >= RightBottom.X && test.Y >= RightBottom.Y)

{

RightBottom = test;

}

}

public int GetLeftTopX()

{

List<int> x = new List<int>();

x.Add(LeftTop.X);

x.Add(RightTop.X);

x.Add(LeftBottom.X);

x.Add(RightBottom.X);

return x.Min();

}

public int GetLeftTopY()

{

List<int> y = new List<int>();

y.Add(LeftTop.Y);

y.Add(RightTop.Y);

y.Add(LeftBottom.Y);

y.Add(RightBottom.Y);

return y.Min();

}

public int GetRightBottomX()

{

List<int> x = new List<int>();

x.Add(LeftTop.X);

x.Add(RightTop.X);

x.Add(LeftBottom.X);

x.Add(RightBottom.X);

return x.Max();

}

public int GetRightBottomY()

{

List<int> y = new List<int>();

y.Add(LeftTop.Y);

y.Add(RightTop.Y);

y.Add(LeftBottom.Y);

y.Add(RightBottom.Y);

return y.Max();

}

public override string ToString()

{

return string.Format("FingerRectangle: LeftTop={0}, RightTop={1}, LeftBottom={2}, RightBottom={3}",

LeftTop, RightTop, LeftBottom, RightBottom);

}

}

using System;

public static class GestureNumberHelper

{

public static string ToString(int gestureNumber) {

switch (gestureNumber) {

case (int) GestureNumber.NOTHING\_HAS\_DETECTED:

return "Nothing has detected.";

case (int) GestureNumber.THUMB\_INDEX\_FINGER:

return "Thumb + index finger";

case (int) GestureNumber.THUMB\_MIDDLE\_FINGER:

return "Thumb + middle finger";

case (int) GestureNumber.THUMB\_PINKY:

return "Thumb + pinky finger";

case (int) GestureNumber.THUMB\_RING\_FINGER:

return "Thumb + ring finger";

}

return "Nothing has detected.";

}

}

using System;

using System.Drawing;

using System.Collections.Generic;

using System.Linq;

public class Gesture

{

List<DatasetItem>[] \_fingers;

DateTime \_timeStamp;

public Rectangle[] FingerRectangles;

public Gesture ()

{

\_timeStamp = DateTime.Now;

FingerRectangles = new Rectangle[5];

}

public int CheckGesture(List<DatasetItem>[] fingers)

{

if (\_timeStamp == DateTime.Now)

return (int) GestureNumber.NOTHING\_HAS\_DETECTED;

// Обновляем время срабатывания

\_timeStamp = DateTime.Now;

\_fingers = fingers;

for (int i = 0; i < \_fingers.Length; i++)

{

var item = \_fingers[i];

Rectangle ThumbRect = new Rectangle((int) item.Min(a => a.X),

(int) item.Min(a => a.Y),

(int) item.Max(a => a.X) - (int) item.Min(a => a.X),

(int) item.Max(a => a.Y) - (int) item.Min(a => a.Y));

FingerRectangles[i] = ThumbRect;

}

for (int i = 1; i < \_fingers.Length; i++) {

if (GetDistance(Center(FingerRectangles[(int)FingersName.THUMB]),

Center(FingerRectangles[i])) <= 50)

return i;

}

return (int) GestureNumber.NOTHING\_HAS\_DETECTED;

}

int GetDistance(Point x, Point y)

{

return (int) Math.Sqrt((x.X - y.X) \* (x.X - y.X) + (x.Y - y.Y) \* (x.Y - y.Y));

}

public Point Center(Rectangle rect)

{

return new Point(rect.Left + rect.Width/2,

rect.Top + rect.Height / 2);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

/// <summary>

/// Structure to define HSB.

/// </summary>

public struct HSB

{

/// <summary>

/// Gets an empty HSB structure;

/// </summary>

public static readonly HSB Empty = new HSB();

private double hue;

private double saturation;

private double brightness;

public static bool operator ==(HSB item1, HSB item2)

{

return (

item1.Hue == item2.Hue

&& item1.Saturation == item2.Saturation

&& item1.Brightness == item2.Brightness

);

}

public static bool operator !=(HSB item1, HSB item2)

{

return (

item1.Hue != item2.Hue

|| item1.Saturation != item2.Saturation

|| item1.Brightness != item2.Brightness

);

}

/// <summary>

/// Gets or sets the hue component.

/// </summary>

public double Hue

{

get

{

return hue;

}

set

{

hue = (value > 360) ? 360 : ((value < 0) ? 0 : value);

}

}

/// <summary>

/// Gets or sets saturation component.

/// </summary>

public double Saturation

{

get

{

return saturation;

}

set

{

saturation = (value > 1) ? 1 : ((value < 0) ? 0 : value);

}

}

/// <summary>

/// Gets or sets the brightness component.

/// </summary>

public double Brightness

{

get

{

return brightness;

}

set

{

brightness = (value > 1) ? 1 : ((value < 0) ? 0 : value);

}

}

/// <summary>

/// Creates an instance of a HSB structure.

/// </summary>

/// <param name="h">Hue value.</param>

/// <param name="s">Saturation value.</param>

/// <param name="b">Brightness value.</param>

public HSB(double h, double s, double b)

{

hue = (h > 360) ? 360 : ((h < 0) ? 0 : h);

saturation = (s > 1) ? 1 : ((s < 0) ? 0 : s);

brightness = (b > 1) ? 1 : ((b < 0) ? 0 : b);

}

public override bool Equals(Object obj)

{

if (obj == null || GetType() != obj.GetType()) return false;

return (this == (HSB)obj);

}

public override int GetHashCode()

{

return Hue.GetHashCode() ^ Saturation.GetHashCode() ^

Brightness.GetHashCode();

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

/// <summary>

/// RGB structure.

/// </summary>

public struct RGB

{

/// <summary>

/// Gets an empty RGB structure;

/// </summary>

public static readonly RGB Empty = new RGB();

private int red;

private int green;

private int blue;

public static bool operator ==(RGB item1, RGB item2)

{

return (

item1.Red == item2.Red

&& item1.Green == item2.Green

&& item1.Blue == item2.Blue

);

}

public static bool operator !=(RGB item1, RGB item2)

{

return (

item1.Red != item2.Red

|| item1.Green != item2.Green

|| item1.Blue != item2.Blue

);

}

/// <summary>

/// Gets or sets red value.

/// </summary>

public int Red

{

get

{

return red;

}

set

{

red = (value > 255) ? 255 : ((value < 0) ? 0 : value);

}

}

/// <summary>

/// Gets or sets red value.

/// </summary>

public int Green

{

get

{

return green;

}

set

{

green = (value > 255) ? 255 : ((value < 0) ? 0 : value);

}

}

/// <summary>

/// Gets or sets red value.

/// </summary>

public int Blue

{

get

{

return blue;

}

set

{

blue = (value > 255) ? 255 : ((value < 0) ? 0 : value);

}

}

public RGB(int R, int G, int B)

{

this.red = (R > 255) ? 255 : ((R < 0) ? 0 : R);

this.green = (G > 255) ? 255 : ((G < 0) ? 0 : G);

this.blue = (B > 255) ? 255 : ((B < 0) ? 0 : B);

}

public override bool Equals(Object obj)

{

if (obj == null || GetType() != obj.GetType()) return false;

return (this == (RGB)obj);

}

public override int GetHashCode()

{

return Red.GetHashCode() ^ Green.GetHashCode() ^ Blue.GetHashCode();

}

}

#### Листинг (WebCamCapture.dll)

using System;

using System.Collections;

using System.ComponentModel;

using System.Drawing;

using System.Data;

using System.Windows.Forms;

using System.Runtime.InteropServices;

namespace WebCam\_Capture

{

/// <summary>

/// Summary description for UserControl1.

/// </summary>

[System.Drawing.ToolboxBitmap(typeof(WebCamCapture), "CAMERA.ICO")] // toolbox bitmap

[Designer("Sytem.Windows.Forms.Design.ParentControlDesigner,System.Design", typeof(System.ComponentModel.Design.IDesigner))] // make composite

public class WebCamCapture : System.Windows.Forms.UserControl

{

private System.ComponentModel.IContainer components; // ??

private System.Windows.Forms.Timer timer1; // ??

// property variables

private int m\_TimeToCapture\_milliseconds = 100;

private int m\_Width = 320;

private int m\_Height = 240;

private int mCapHwnd;

private ulong m\_FrameNumber = 0;

// global variables to make the video capture go faster

private WebCam\_Capture.WebcamEventArgs x = new WebCam\_Capture.WebcamEventArgs();

private IDataObject tempObj;

private System.Drawing.Image tempImg;

private bool bStopped = true;

// event delegate Îáðàáîò÷èê ñîáûòèé ñ êàìåðû

public delegate void WebCamEventHandler (object source, WebCam\_Capture.WebcamEventArgs e);

// fired when a new image is captured

public event WebCamEventHandler ImageCaptured;

#region API Declarations

[DllImport("user32", EntryPoint="SendMessage")]

public static extern int SendMessage(int hWnd, uint Msg, int wParam, int lParam);

[DllImport("avicap32.dll", EntryPoint="capCreateCaptureWindowA")]

public static extern int capCreateCaptureWindowA(string lpszWindowName, int dwStyle, int X, int Y, int nWidth, int nHeight, int hwndParent, int nID);

[DllImport("user32", EntryPoint="OpenClipboard")]

public static extern int OpenClipboard(int hWnd);

[DllImport("user32", EntryPoint="EmptyClipboard")]

public static extern int EmptyClipboard();

[DllImport("user32", EntryPoint="CloseClipboard")]

public static extern int CloseClipboard();

#endregion

#region API Constants

public const int WM\_USER = 1024;

public const int WM\_CAP\_CONNECT = 1034;

public const int WM\_CAP\_DISCONNECT = 1035;

public const int WM\_CAP\_GET\_FRAME = 1084;

public const int WM\_CAP\_COPY = 1054;

public const int WM\_CAP\_START = WM\_USER;

public const int WM\_CAP\_DLG\_VIDEOFORMAT = WM\_CAP\_START + 41;

public const int WM\_CAP\_DLG\_VIDEOSOURCE = WM\_CAP\_START + 42;

public const int WM\_CAP\_DLG\_VIDEODISPLAY = WM\_CAP\_START + 43;

public const int WM\_CAP\_GET\_VIDEOFORMAT = WM\_CAP\_START + 44;

public const int WM\_CAP\_SET\_VIDEOFORMAT = WM\_CAP\_START + 45;

public const int WM\_CAP\_DLG\_VIDEOCOMPRESSION = WM\_CAP\_START + 46;

public const int WM\_CAP\_SET\_PREVIEW = WM\_CAP\_START + 50;

#endregion

#region NOTES

/\*

\* If you want to allow the user to change the display size and

\* color format of the video capture, call:

\* SendMessage (mCapHwnd, WM\_CAP\_DLG\_VIDEOFORMAT, 0, 0);

\* You will need to requery the capture device to get the new settings

\*/

#endregion

public WebCamCapture()

{

// This call is required by the Windows.Forms Form Designer.

InitializeComponent();

}

/// <summary>

/// Override the class's finalize method, so we can stop

/// the video capture on exit

/// </summary>

~WebCamCapture()

{

this.Stop();

}

/// <summary>

/// Clean up any resources being used.

/// </summary>

protected override void Dispose( bool disposing )

{

if( disposing )

{

if( components != null )

components.Dispose();

}

base.Dispose( disposing );

}

#region Component Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

private void InitializeComponent()

{

this.components = new System.ComponentModel.Container();

this.timer1 = new System.Windows.Forms.Timer(this.components);

this.SuspendLayout();

//

// timer1

//

this.timer1.Tick += new System.EventHandler(this.timer1\_Tick);

//

// WebCamCapture

//

this.Name = "WebCamCapture";

this.Size = new System.Drawing.Size(342, 252);

this.Load += new System.EventHandler(this.WebCamCapture\_Load);

this.ResumeLayout(false);

}

#endregion

#region Control Properties

/// <summary>

/// The time intervale between frame captures

/// </summary>

public int TimeToCapture\_milliseconds

{

get

{ return m\_TimeToCapture\_milliseconds; }

set

{ m\_TimeToCapture\_milliseconds = value; }

}

/// <summary>

/// The height of the video capture image

/// </summary>

public int CaptureHeight

{

get

{ return m\_Height; }

set

{ m\_Height = value; }

}

/// <summary>

/// The width of the video capture image

/// </summary>

public int CaptureWidth

{

get

{ return m\_Width; }

set

{ m\_Width = value; }

}

/// <summary>

/// The sequence number to start at for the frame number. OPTIONAL

/// </summary>

public ulong FrameNumber

{

get

{ return m\_FrameNumber; }

set

{ m\_FrameNumber = value; }

}

#endregion

#region Start and Stop Capture Functions

/// <summary>

/// Starts the video capture

/// </summary>

/// <param name="FrameNumber">the frame number to start at.

/// Set to 0 to let the control allocate the frame number</param>

public void Start(ulong FrameNum)

{

try

{

// for safety, call stop, just in case we are already running

this.Stop();

// setup a capture window

mCapHwnd = capCreateCaptureWindowA("WebCap", 0, 0, 0, m\_Width, m\_Height, this.Handle.ToInt32(), 0);

// connect to the capture device

Application.DoEvents();

SendMessage(mCapHwnd, WM\_CAP\_CONNECT, 0, 0);

SendMessage(mCapHwnd, WM\_CAP\_SET\_PREVIEW, 0, 0);

// set the frame number

m\_FrameNumber = FrameNum;

// set the timer information

this.timer1.Interval = m\_TimeToCapture\_milliseconds;

bStopped = false;

this.timer1.Start();

}

catch (Exception excep)

{

MessageBox.Show("An error ocurred while starting the video capture. Check that your webcamera is connected properly and turned on.\r\n\n" + excep.Message);

this.Stop();

}

}

/// <summary>

/// Stops the video capture

/// </summary>

public void Stop()

{

try

{

// stop the timer

bStopped = true;

this.timer1.Stop();

// disconnect from the video source

Application.DoEvents();

SendMessage(mCapHwnd, WM\_CAP\_DISCONNECT, 0, 0);

}

catch (Exception excep)

{ // don't raise an error here.

}

}

#endregion

#region Video Capture Code

/// <summary>

/// Capture the next frame from the video feed

/// </summary>

private void timer1\_Tick(object sender, System.EventArgs e)

{

try

{

// pause the timer

this.timer1.Stop();

// get the next frame;

SendMessage(mCapHwnd, WM\_CAP\_GET\_FRAME, 0, 0);

// copy the frame to the clipboard

SendMessage(mCapHwnd, WM\_CAP\_COPY, 0, 0);

// paste the frame into the event args image

if (ImageCaptured != null)

{

// get from the clipboard

tempObj = Clipboard.GetDataObject();

tempImg = (System.Drawing.Bitmap) tempObj.GetData(System.Windows.Forms.DataFormats.Bitmap);

/\*

\* For some reason, the API is not resizing the video

\* feed to the width and height provided when the video

\* feed was started, so we must resize the image here

\*/

x.WebCamImage = tempImg.GetThumbnailImage(m\_Width, m\_Height, null, System.IntPtr.Zero);

// raise the event

this.ImageCaptured(this, x);

}

// restart the timer

Application.DoEvents();

if (! bStopped)

this.timer1.Start();

}

catch (Exception excep)

{

MessageBox.Show("An error ocurred while capturing the video image. The video capture will now be terminated.\r\n\n" + excep.Message);

this.Stop(); // stop the process

}

}

#endregion

private void WebCamCapture\_Load(object sender, EventArgs e)

{

}

}

}

using System;

namespace WebCam\_Capture

{

/// <summary>

/// EventArgs for the webcam control

/// </summary>

public class WebcamEventArgs : System.EventArgs

{

private System.Drawing.Image m\_Image;

private ulong m\_FrameNumber = 0;

public WebcamEventArgs()

{

}

/// <summary>

/// WebCamImage

/// This is the image returned by the web camera capture

/// </summary>

public System.Drawing.Image WebCamImage

{

get

{ return m\_Image; }

set

{ m\_Image = value; }

}

/// <summary>

/// FrameNumber

/// Holds the sequence number of the frame capture

/// </summary>

public ulong FrameNumber

{

get

{ return m\_FrameNumber; }

set

{ m\_FrameNumber = value; }

}

}

}

#### Листинг (Paint with WebCam)

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.Runtime.InteropServices;

namespace Paint

{

public partial class Form1 : Form

{

// Paint sector

private bool Brush = true; //Uses either Brush or Eraser. Default is Brush

private Shapes DrawingShapes = new Shapes(); //Stores all the drawing data

private bool IsPainting = false; //Is the mouse currently down (PAINTING)

private bool IsEraseing = false; //Is the mouse currently down (ERASEING)

private Point LastPos = new Point(0, 0); //Last Position, used to cut down on repative data.

private Color CurrentColour = Color.Black; //Deafult Colour

private float CurrentWidth = 10; //Deafult Pen width

private int ShapeNum = 0; //record the shapes so they can be drawn sepratley.

private Point MouseLoc = new Point(0, 0); //Record the mouse position

private bool IsMouseing = false; //Draw the mouse?

// WebCamera sector

private WebCam\_Capture.WebCamCapture WebCamCapture;

private Gesture GestureListener;

private int GestureCode;

private int LastGestureCode;

private Point FingerCenter;

private MouseOperations CatchedMouse;

private DateTime timeStamp;

public Form1()

{

InitializeComponent();

//Set Double Buffering

panel1.GetType().GetMethod("SetStyle", System.Reflection.BindingFlags.Instance | System.Reflection.BindingFlags.NonPublic).Invoke(panel1, new object[] { System.Windows.Forms.ControlStyles.UserPaint | System.Windows.Forms.ControlStyles.AllPaintingInWmPaint | System.Windows.Forms.ControlStyles.DoubleBuffer, true });

}

private void Form1\_Load(object sender, System.EventArgs e)

{

// set the image capture size

this.WebCamCapture.CaptureHeight = 450;

this.WebCamCapture.CaptureWidth = 600;

// change the capture time frame

this.WebCamCapture.TimeToCapture\_milliseconds = 100;

GestureListener = new Gesture();

FingerCenter = new Point();

timeStamp = DateTime.Now;

GestureCode = (int)GestureNumber.NOTHING\_HAS\_DETECTED;

LastGestureCode = (int)GestureNumber.NOTHING\_HAS\_DETECTED;

// start the video capture. let the control handle the

// frame numbers.

this.WebCamCapture.Start(0);

}

private void Form1\_Closing(object sender, System.ComponentModel.CancelEventArgs e)

{

// stop the video capture

this.WebCamCapture.Stop();

}

/// <summary>

/// An image was capture

/// </summary>

/// <param name="source">control raising the event</param>

/// <param name="e">WebCamEventArgs</param>

private void WebCamCapture\_ImageCaptured(object source, WebCam\_Capture.WebcamEventArgs e)

{

try

{

BitMap32 bitmapImage = new BitMap32(e.WebCamImage);

e.WebCamImage.Save("log.jpg");

bitmapImage.HSBFindColoredPlots();

GestureListener.ToString();

List<DatasetItem>[] Fingers = bitmapImage.GetFingers();

for (int i = 0; i < Fingers.Length; i++)

{

if (Fingers[i] == null || Fingers[i].Count == 0)

{

return;

}

}

GestureCode = GestureListener.CheckGesture(Fingers);

System.IO.StreamWriter file = new System.IO.StreamWriter("log.txt", true);

file.WriteLine(DateTime.Now + " " + GestureNumberHelper.ToString(GestureCode));

file.Close();

this.Cursor = new Cursor(Cursor.Current.Handle);

Point newFingerCenter = GestureListener.Center(GestureListener.FingerRectangles[(int)FingersName.THUMB]);

Point currentCursor = Cursor.Position;

Cursor.Position = new Point(currentCursor.X - (newFingerCenter.X - FingerCenter.X),

currentCursor.Y + (newFingerCenter.Y - FingerCenter.Y));

Cursor.Clip = new Rectangle(this.Location, this.Size);

this.FingerCenter = newFingerCenter;

float width = 200;

float height = 150;

var brush = new SolidBrush(Color.Black);

var image = bitmapImage.ImageWithColoredPlots();

float scale = Math.Min(width / image.Width, height / image.Height);

var bmp = new Bitmap((int)width, (int)height);

var graph = Graphics.FromImage(bmp);

// uncomment for higher quality output

//graph.InterpolationMode = InterpolationMode.High;

//graph.CompositingQuality = CompositingQuality.HighQuality;

//graph.SmoothingMode = SmoothingMode.AntiAlias;

var scaleWidth = (int)(image.Width \* scale);

var scaleHeight = (int)(image.Height \* scale);

graph.FillRectangle(brush, new RectangleF(0, 0, width, height));

graph.DrawImage(image, new Rectangle(((int)width - scaleWidth) / 2, ((int)height - scaleHeight) / 2, scaleWidth, scaleHeight));

pictureBox1.Image = bmp;

//DateTime pasteTimeStamp = timeStamp;

//timeStamp = DateTime.Now;

//if (pasteTimeStamp.Second - timeStamp.Second < 1)

// return;

file = new System.IO.StreamWriter("log.txt", true);

if (GestureCode == (int)GestureNumber.THUMB\_INDEX\_FINGER && GestureCode != LastGestureCode)

{

file.WriteLine(DateTime.Now + "Left mouse down");

if (!Brush)

Brush = true;

MouseOperations.MouseEvent(MouseOperations.MouseEventFlags.LeftDown);

LastGestureCode = GestureCode;

}

if (LastGestureCode == (int)GestureNumber.THUMB\_INDEX\_FINGER

&& GestureCode == (int)GestureNumber.NOTHING\_HAS\_DETECTED)

{

file.WriteLine(DateTime.Now + "Left mouse up");

MouseOperations.MouseEvent(MouseOperations.MouseEventFlags.LeftUp);

LastGestureCode = GestureCode;

}

if (GestureCode == (int)GestureNumber.THUMB\_MIDDLE\_FINGER && GestureCode != LastGestureCode)

{

if (Brush)

Brush = false;

file.WriteLine(DateTime.Now + "Right mouse down");

MouseOperations.MouseEvent(MouseOperations.MouseEventFlags.LeftDown);

LastGestureCode = GestureCode;

}

if (LastGestureCode == (int)GestureNumber.THUMB\_MIDDLE\_FINGER

&& GestureCode == (int)GestureNumber.NOTHING\_HAS\_DETECTED)

{

file.WriteLine(DateTime.Now + "Right mouse up");

MouseOperations.MouseEvent(MouseOperations.MouseEventFlags.LeftUp);

LastGestureCode = GestureCode;

}

if (GestureCode == (int)GestureNumber.THUMB\_RING\_FINGER && GestureCode != LastGestureCode)

{

file.WriteLine(DateTime.Now + "Switch color down");

LastGestureCode = GestureCode;

}

if (LastGestureCode == (int)GestureNumber.THUMB\_RING\_FINGER

&& GestureCode == (int)GestureNumber.NOTHING\_HAS\_DETECTED)

{

file.WriteLine(DateTime.Now + "Switch color up");

Random rnd = new Random();

CurrentColour = Color.FromArgb(rnd.Next(0, 255), rnd.Next(0, 255), rnd.Next(0, 255));

LastGestureCode = GestureCode;

}

file.Close();

//if (GestureCode == (int)GestureNumber.THUMB\_MIDDLE\_FINGER)

// MouseOperations.MouseEvent(MouseOperations.MouseEventFlags.RightDown | MouseOperations.MouseEventFlags.RightUp);

}

catch (Exception)

{

// stop the video capture

this.WebCamCapture.Stop();

// start the video capture. let the control handle the

// frame numbers.

this.WebCamCapture.Start(0);

throw;

}

}

private void panel1\_MouseDown(object sender, System.Windows.Forms.MouseEventArgs e)

{

//If we're painting...

if (Brush)

{

//set it to mouse down, illatrate the shape being drawn and reset the last position

IsPainting = true;

ShapeNum++;

LastPos = new Point(0, 0);

}

//but if we're eraseing...

else

{

IsEraseing = true;

}

}

protected void panel1\_MouseMove(object sender, System.Windows.Forms.MouseEventArgs e)

{

MouseLoc = e.Location;

//PAINTING

if (IsPainting)

{

//check its not at the same place it was last time, saves on recording more data.

if (LastPos != e.Location)

{

//set this position as the last positon

LastPos = e.Location;

//store the position, width, colour and shape relation data

DrawingShapes.NewShape(LastPos, CurrentWidth, CurrentColour, ShapeNum);

}

}

if (IsEraseing)

{

//Remove any point within a certain distance of the mouse

DrawingShapes.RemoveShape(e.Location,10);

}

//refresh the panel so it will be forced to re-draw.

panel1.Refresh();

}

private void panel1\_MouseUp(object sender, System.Windows.Forms.MouseEventArgs e)

{

if (IsPainting)

{

//Finished Painting.

IsPainting = false;

}

if (IsEraseing)

{

//Finished Earsing.

IsEraseing = false;

}

}

//DRAWING FUNCTION

private void panel1\_Paint(object sender, PaintEventArgs e)

{

//Apply a smoothing mode to smooth out the line.

e.Graphics.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.HighQuality;

//DRAW THE LINES

for (int i = 0; i < DrawingShapes.NumberOfShapes()-1; i++)

{

Shape T = DrawingShapes.GetShape(i);

Shape T1 = DrawingShapes.GetShape(i+1);

//make sure shape the two ajoining shape numbers are part of the same shape

if (T.ShapeNumber == T1.ShapeNumber)

{

//create a new pen with its width and colour

Pen p = new Pen(T.Colour, T.Width);

p.StartCap = System.Drawing.Drawing2D.LineCap.Round;

p.EndCap = System.Drawing.Drawing2D.LineCap.Round;

//draw a line between the two ajoining points

e.Graphics.DrawLine(p, T.Location, T1.Location);

//get rid of the pen when finished

p.Dispose();

}

}

//If mouse is on the panel, draw the mouse

if (IsMouseing || GestureCode == (int) GestureNumber.THUMB\_INDEX\_FINGER)

{

e.Graphics.DrawEllipse(new Pen(Color.Aquamarine, 2f), MouseLoc.X - (CurrentWidth / 2), MouseLoc.Y - (CurrentWidth / 2), CurrentWidth, CurrentWidth);

}

}

private void button1\_Click(object sender, EventArgs e)

{

//Go from the BRUSH to the ERASER

Brush = !Brush;

}

private void button2\_Click(object sender, EventArgs e)

{

//Show and Get the result of the colour dialog

DialogResult D = colorDialog1.ShowDialog();

if (D == DialogResult.OK)

{

//Apply the new colour

CurrentColour = colorDialog1.Color;

}

}

private void numericUpDown1\_ValueChanged(object sender, EventArgs e)

{

//Change the width of the pen. Oh and convert it to a float

CurrentWidth = Convert.ToSingle(numericUpDown1.Value);

}

private void button3\_Click(object sender, EventArgs e)

{

//Reset the list, removeing all shapes.

DrawingShapes = new Shapes();

panel1.Refresh();

}

private void panel1\_MouseEnter(object sender, EventArgs e)

{

//Hide the mouse cursor and tell the re-drawing function to draw the mouse

Cursor.Hide();

IsMouseing = true;

}

private void panel1\_MouseLeave(object sender, EventArgs e)

{

//show the mouse, tell the re-drawing function to stop drawing it and force the panel to re-draw.

Cursor.Show();

IsMouseing = false;

panel1.Refresh();

}

}

public class Shape

{

public Point Location; //position of the point

public float Width; //width of the line

public Color Colour; //colour of the line

public int ShapeNumber; //part of which shape it belongs to

//CONSTRUCTOR

public Shape(Point L, float W, Color C, int S)

{

Location = L; //Stores the Location

Width = W; //Stores the width

Colour = C; //Stores the colour

ShapeNumber = S; //Stores the shape number

}

}

public class Shapes

{

private List<Shape> \_Shapes; //Stores all the shapes

public Shapes()

{

\_Shapes = new List<Shape>();

}

//Returns the number of shapes being stored.

public int NumberOfShapes()

{

return \_Shapes.Count;

}

//Add a shape to the database, recording its position, width, colour and shape relation information

public void NewShape(Point L, float W, Color C, int S)

{

\_Shapes.Add(new Shape(L,W,C,S));

}

//returns a shape of the requested data.

public Shape GetShape(int Index)

{

return \_Shapes[Index];

}

//Removes any point data within a certain threshold of a point.

public void RemoveShape(Point L, float threshold)

{

for (int i = 0; i < \_Shapes.Count; i++)

{

//Finds if a point is within a certain distance of the point to remove.

if ((Math.Abs(L.X - \_Shapes[i].Location.X) < threshold) &&(Math.Abs(L.Y - \_Shapes[i].Location.Y)< threshold))

{

//removes all data for that number

\_Shapes.RemoveAt(i);

//goes through the rest of the data and adds an extra 1 to defined them as a seprate shape and shuffles on the effect.

for (int n = i; n < \_Shapes.Count; n++)

{

\_Shapes[n].ShapeNumber += 1;

}

//Go back a step so we dont miss a point.

i -= 1;

}

}

}

}

public class MouseOperations

{

[Flags]

public enum MouseEventFlags

{

LeftDown = 0x02,

LeftUp = 0x04,

MiddleDown = 0x20,

MiddleUp = 0x40,

Move = 0x01,

Absolute = 0x8000,

RightDown = 0x08,

RightUp = 0x10

}

[DllImport("user32.dll", EntryPoint = "SetCursorPos")]

[return: MarshalAs(UnmanagedType.Bool)]

private static extern bool SetCursorPos(int X, int Y);

[DllImport("user32.dll")]

[return: MarshalAs(UnmanagedType.Bool)]

private static extern bool GetCursorPos(out MousePoint lpMousePoint);

[DllImport("user32.dll")]

private static extern void mouse\_event(int dwFlags, int dx, int dy, int dwData, int dwExtraInfo);

public static void SetCursorPosition(int X, int Y)

{

SetCursorPos(X, Y);

}

public static void SetCursorPosition(MousePoint point)

{

SetCursorPos(point.X, point.Y);

}

public static MousePoint GetCursorPosition()

{

MousePoint currentMousePoint;

var gotPoint = GetCursorPos(out currentMousePoint);

if (!gotPoint) { currentMousePoint = new MousePoint(0, 0); }

return currentMousePoint;

}

public static void MouseEvent(MouseEventFlags value)

{

MousePoint position = GetCursorPosition();

mouse\_event

((int)value,

position.X,

position.Y,

0,

0)

;

}

[StructLayout(LayoutKind.Sequential)]

public struct MousePoint

{

public int X;

public int Y;

public MousePoint(int x, int y)

{

X = x;

Y = y;

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows.Forms;

namespace Paint

{

static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

#### Список литературы

1. Рихтер Дж. - CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C# (Мастер-класс) – 2012
2. URL: Обзор алгоритмов кластеризации данных https://habrahabr.ru/post/101338/
3. URL: http://www.codeproject.com/Articles/19045/Manipulating-colors-in-NET-Part
4. URL: https://www.yzuzun.com/2015/07/dbscan-clustering-algorithm-and-c-implementation/
5. URL: Путь радуги: Алгоритм распознавания движений пальцев рук на основе цветовой диффференциации - http://shamansir.github.io/blog/ru/articles/nijiato-detection-of-fingers-motion-algorithm/