Лабораторная работа №1

Основы работы с библиотекой компьютерного зрения OpenCV

1. Общие сведения

В современном мире мы часто сталкиваемся с приложениями, которые используют камеры. В некоторых цифровых фотоаппаратах доступна функция распознавания лиц. В кино мы видим работу с программами, которые с легкостью могут распознать личность по фотографии или снимку с камеры видеонаблюдения. Существуют программы, которые могут распознавать текст, представленный на фотографии. Это является примером компьютерного зрения.

Одной из самых популярных библиотек компьютерного зрения является библиотека с открытым исходным кодом «OpenCV». Библиотека постоянно обновляется и увеличивает количество включенных в нее функций и алгоритмов.

OpenCV распространяется по лицензии BSD (условие использования: наличие в сопровождающих материалах копии лицензии OpenCV), следовательно, ее можно использовать для небольших проектов с исходным кодом и для больших коммерческих проектов. Библиотека поддерживает работу с различными операционными системами: MS Windows, Linux, Mac, Android, iOs.

Проект постоянно обновляется. Для проверки работоспособности конфигурации библиотеки она загружается на устройства с той или иной операционной системой, после чего проводится реальное тестирование.

Библиотека OpenCV имеет несколько различных модулей, каждый из которых решает различные задачи. Модули можно разделить на 4 группы.

- Модули core, highgui, реализующие базовую функциональность (базовые структуры, математические функции, генераторы случайных чисел, линейная алгебра, быстрое преобразование Фурье, ввод/вывод изображений и видео, ввод/вывод в форматах XML, YAML и др.).
- Модули imgproc, features2d для обработки изображений (фильтрация, геометрические преобразования, преобразование цветовых пространств, сегментация, обнаружение особых точек и ребер, контурный анализ и др).
- Модули video, objdetect, calib3d (калибровка камеры, анализ движения и отслеживание объектов, вычисление положения в пространстве, построение карты глубины, детектирование объектов, оптический поток).
- Модуль ml, реализующий алгоритмы машинного обучения (метод ближайших соседей, наивный байесовский классификатор, деревья решений, бустинг, градиентный бустинг деревьев решений, случайный лес, машина опорных векторов, нейронные сети и др.).

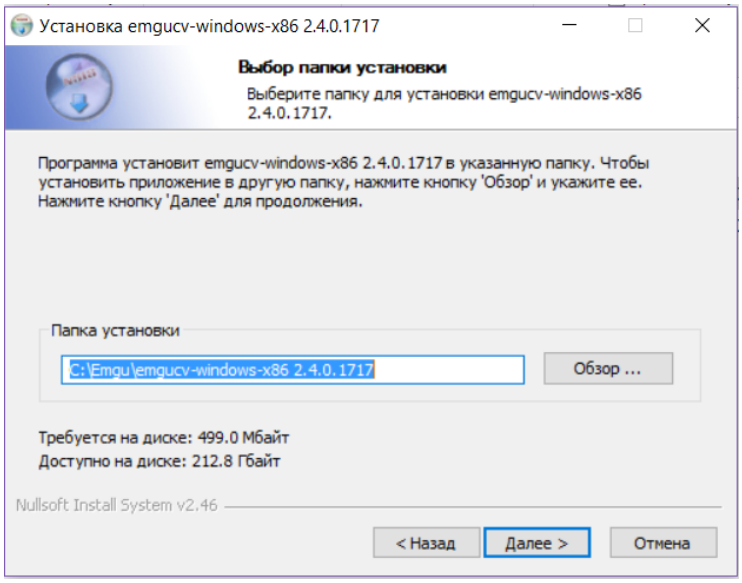
Одним из важных качеств данной библиотеки является поддержка работы с XML-файлами.

2. Подготовка к использованию библиотеки OpenCV (EmguCV for C#)

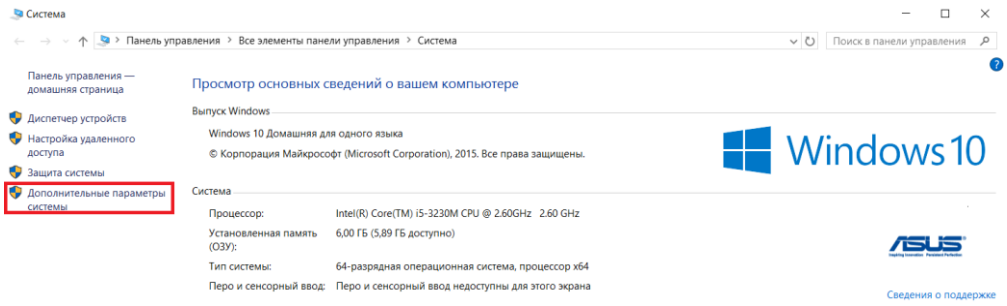
Emgu CV является кросс-платформенным .Net дополнением для библиотеки OpenCV для обработки изображений. Разработано для работы с .NET совместимыми языками, такими как C #, VB, VC ++, IronPython и т.д., может быть использовано в Visual Studio, Xamarin, работает с Windows, Linux, Mac OS X, IOS, Android и Windows Phone.

1. Скачать и установить среду программирования Visual Studio версии 2013 года и выше.

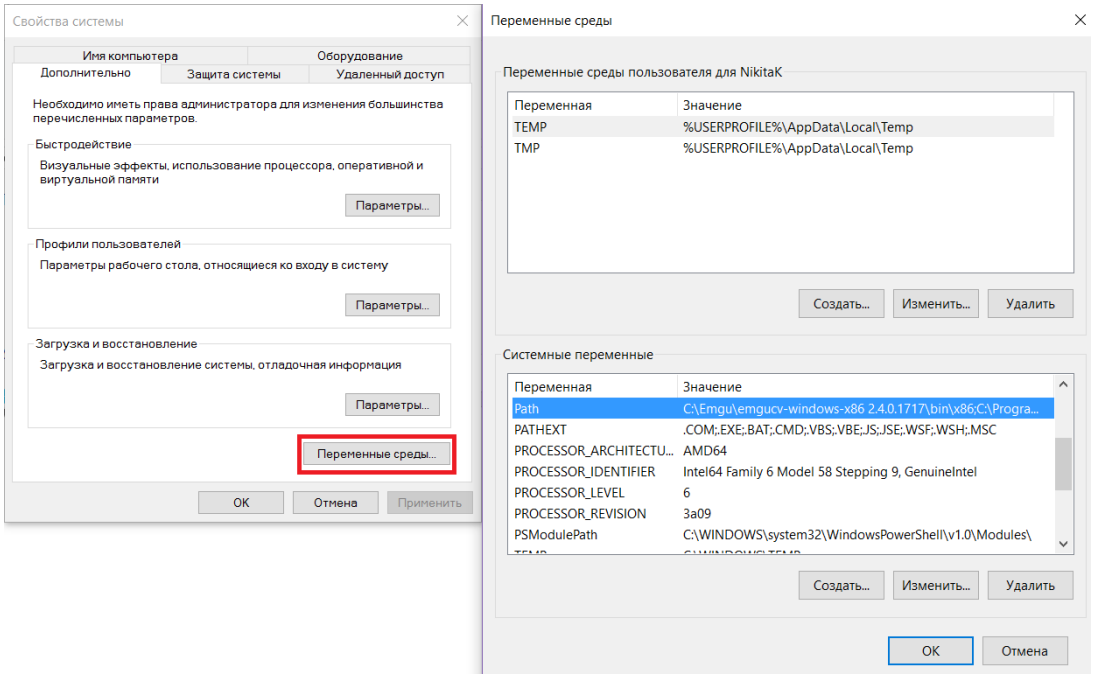
2. Скачать и установить библиотеку Emgu CV (наиболее стабильна версия 2.4.0):
   Запустить установщик, на последующих шагах оставить все без изменений.



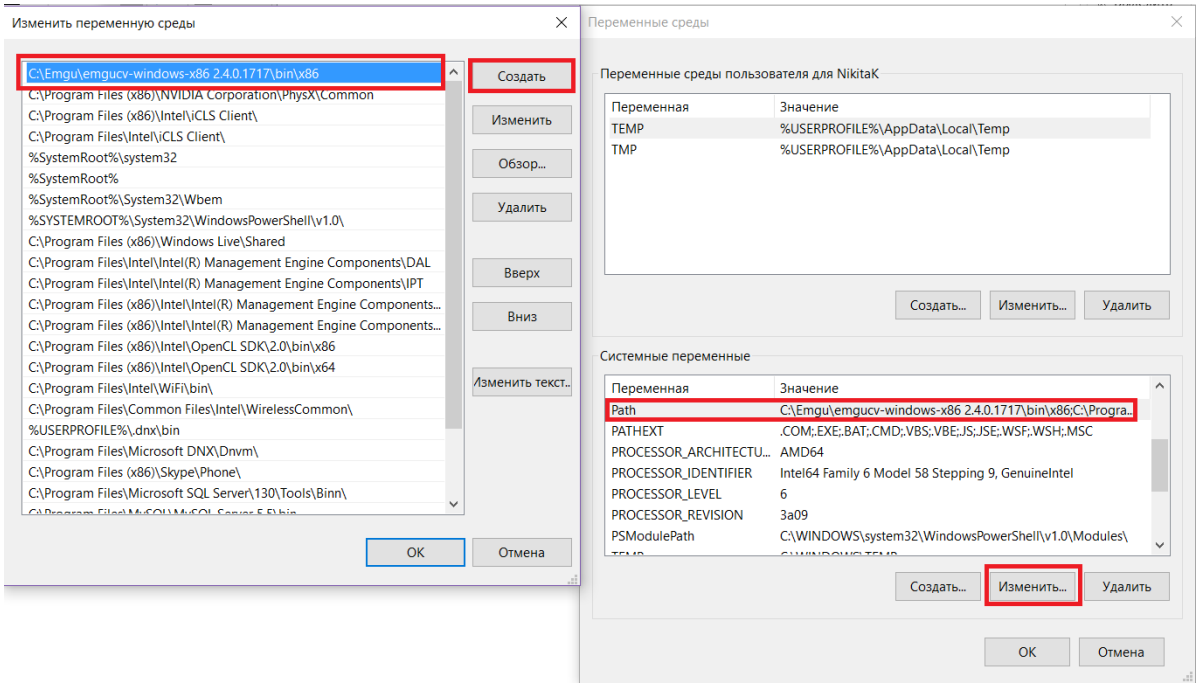
3. Изменяем переменную окружения
   a. «Этот компьютер» → (нажатие правой кнопки мыши) «Свойства» → «Дополнительные параметры системы»



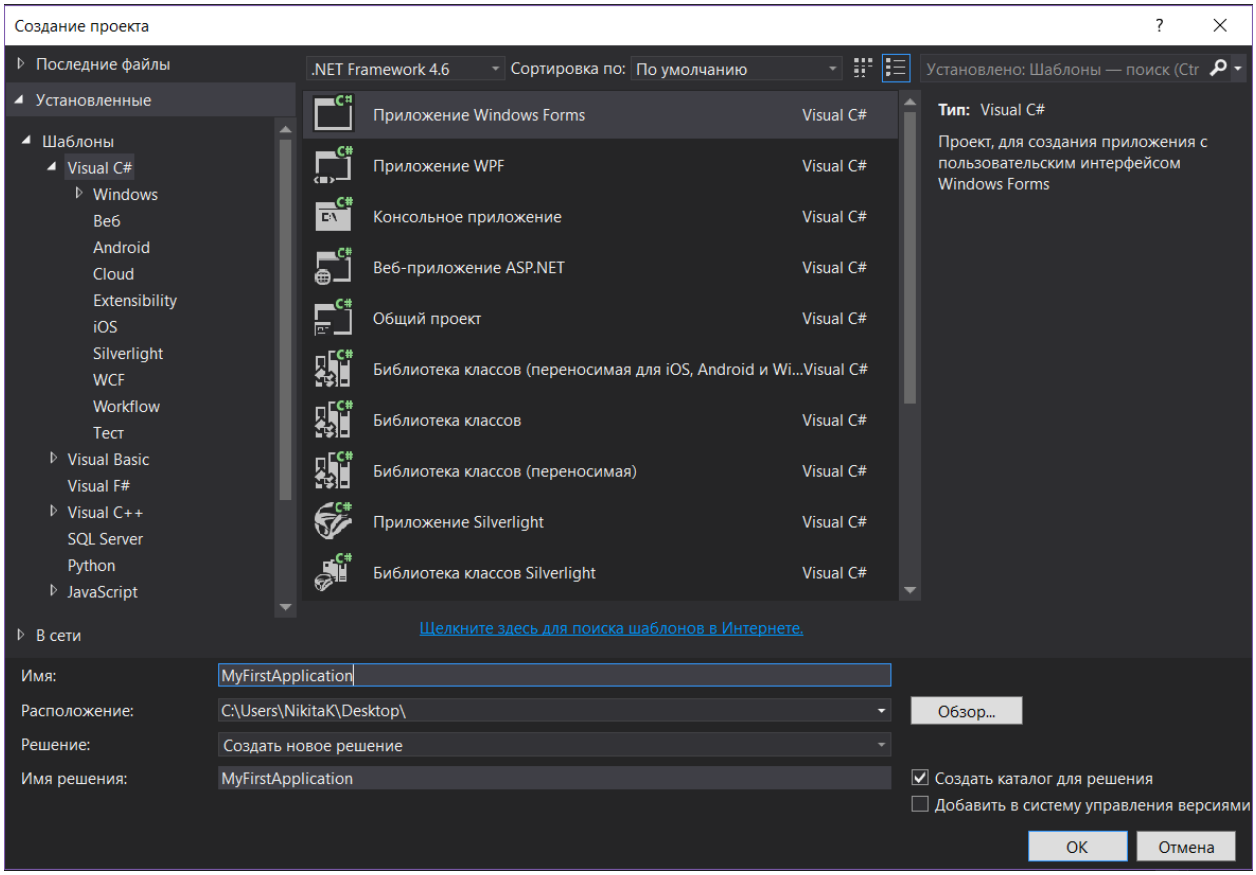
   b. Вкладка «Дополнительно» → «Переменные среды…»

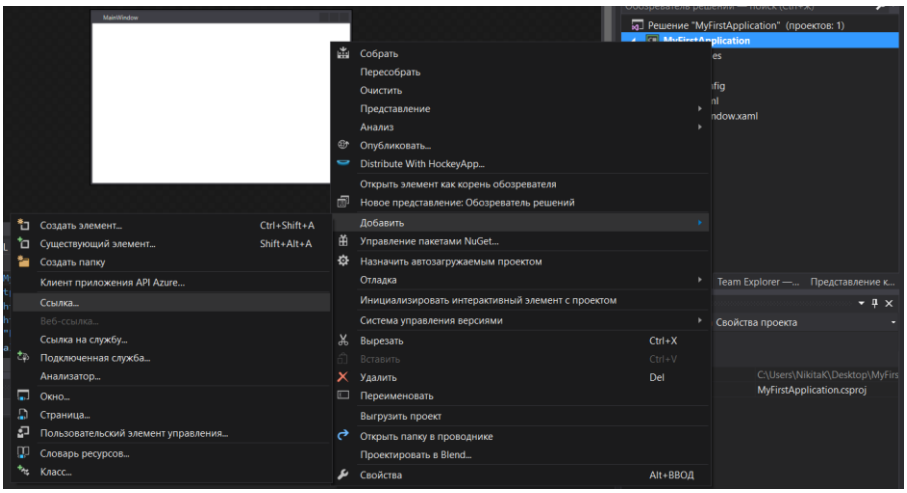


c. Изменить системную переменную окружения «Path», создав новый элемент (Добавить сроку: C:\Emgu\emgucv-windows-x86 2.4.0.1717\bin\x86)

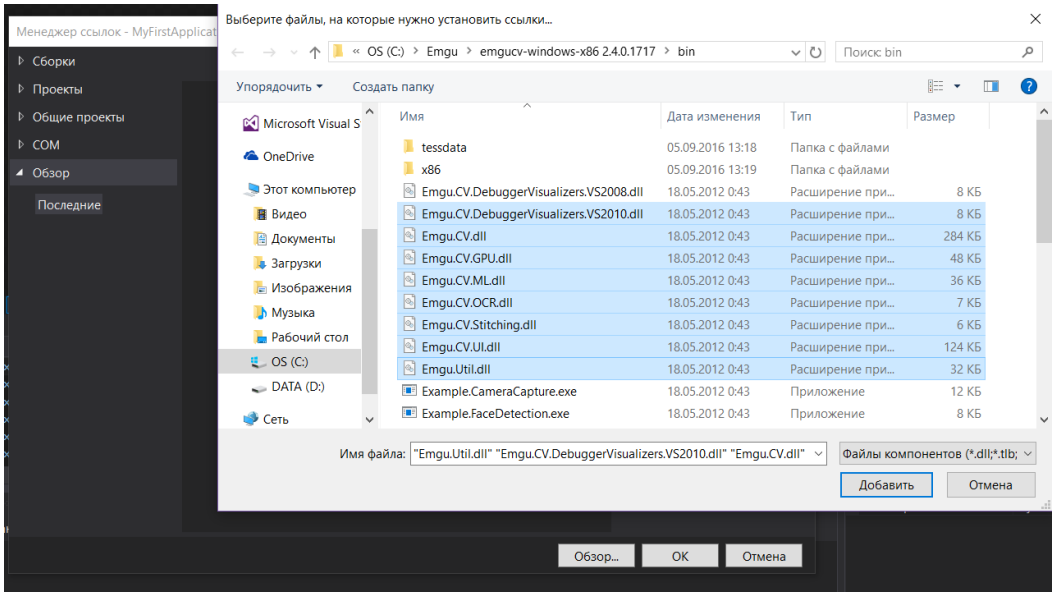


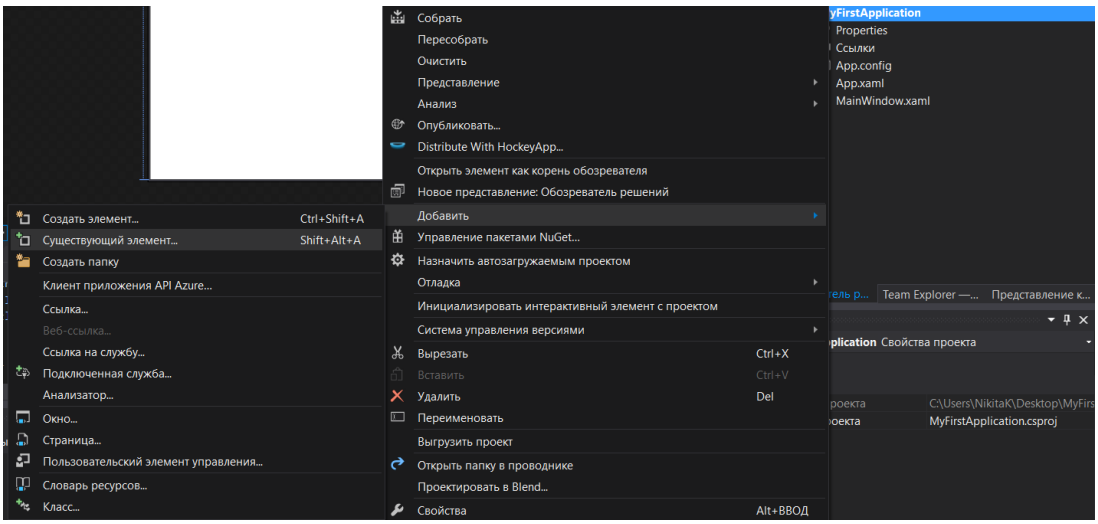
4. Открываем VS и создаем новое приложение Windows Forms



5. Добавить в проект ссылки
   a. 

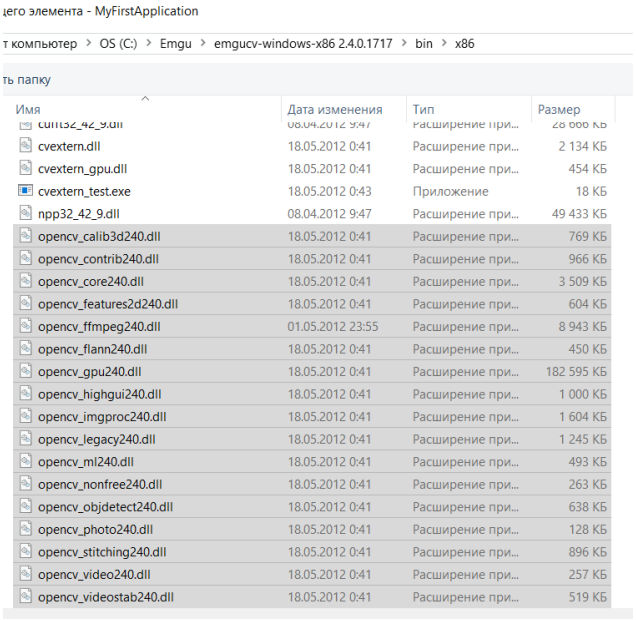
   b. «Обзор…» → C:\Emgu\emgucv-windows-x86 2.4.0.1717\bin (Добавляем 8 dll)



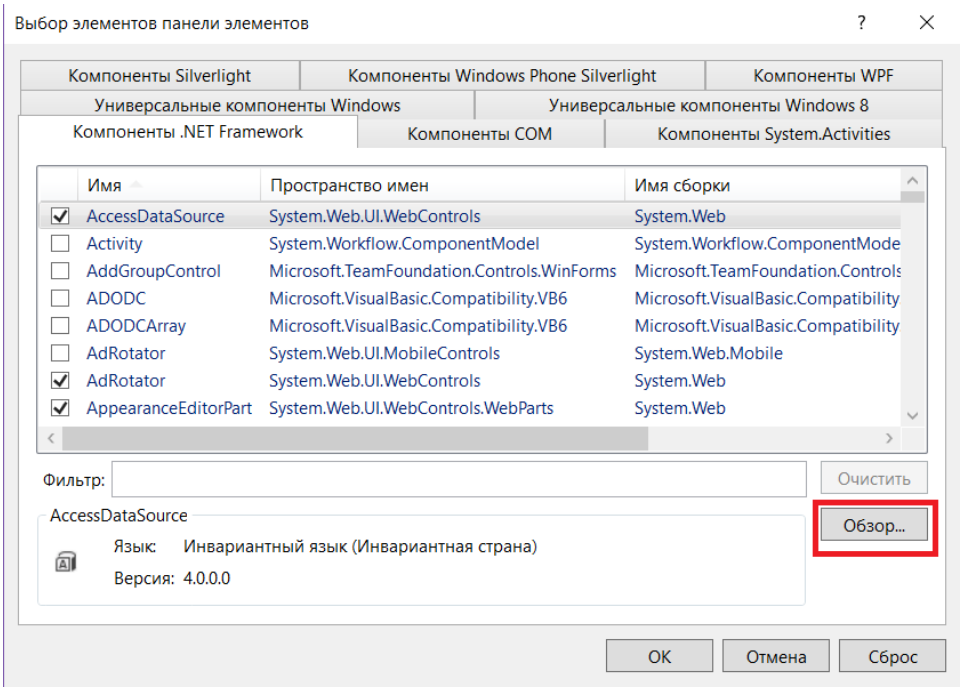
6. Добавляем существующие элементы
   a. 

   b. C:\Emgu\emgucv-windows-x86 2.4.0.1717\bin\x86

c. Добавить 17 dll, начинающихся с «opencv_» (файлы .exe и .dll)

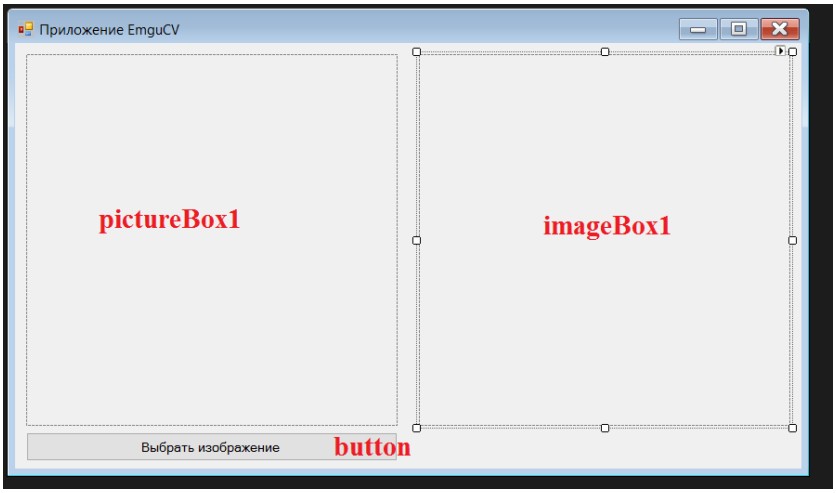


7. Добавить новые компоненты панели элементов (Правая кнопка мыши) → «Выбрать элементы» → «Компоненты .NET Framework» → «Обзор…» → C:\Emgu\emgucv-windows-x86 2.4.0.1717\bin → Emgu.CV.UI.dll



3. Создание первого приложения с использованием EmguCV

1. Создать окно приложения, добавив следующие элементы: pictureBox, imageBox и button



2. Подключаем следующие библиотеки:

```
using Emgu.CV;
using Emgu.CV.CvEnum;
using Emgu.CV.Structure;
using Emgu.CV.Cvb;
```

3. Создаем обработчик нажатия на кнопку «Выбрать изображение»

```
private void button1_Click(object sender, EventArgs e)
{
    OpenFileDialog ok = new OpenFileDialog();
    Image img = null;
    string name = "";
    if(ok.ShowDialog()==System.Windows.Forms.DialogResult.OK)
    {
        img = Image.FromFile(ok.FileName);
        name = ok.FileName;
        pictureBox1.Image = img;
    }
    function(name);
}
```

4. Создаем функцию обработки изображения и поиска элементов лица
   a. Подключаем файлы .xml, которые содержат элементы распознавания лица

```
HaarCascade faceCascade = new HaarCascade(@"C:\Emgu\emgucv-windows-x86 2.4.0.1717\bin\haarcascade_frontalface_default.xml");
HaarCascade eyeCascade = new HaarCascade(@"C:\Emgu\emgucv-windows-x86 2.4.0.1717\bin\haarcascade_eye.xml");
```

   b. Добавляем код обработки и запускаем приложение

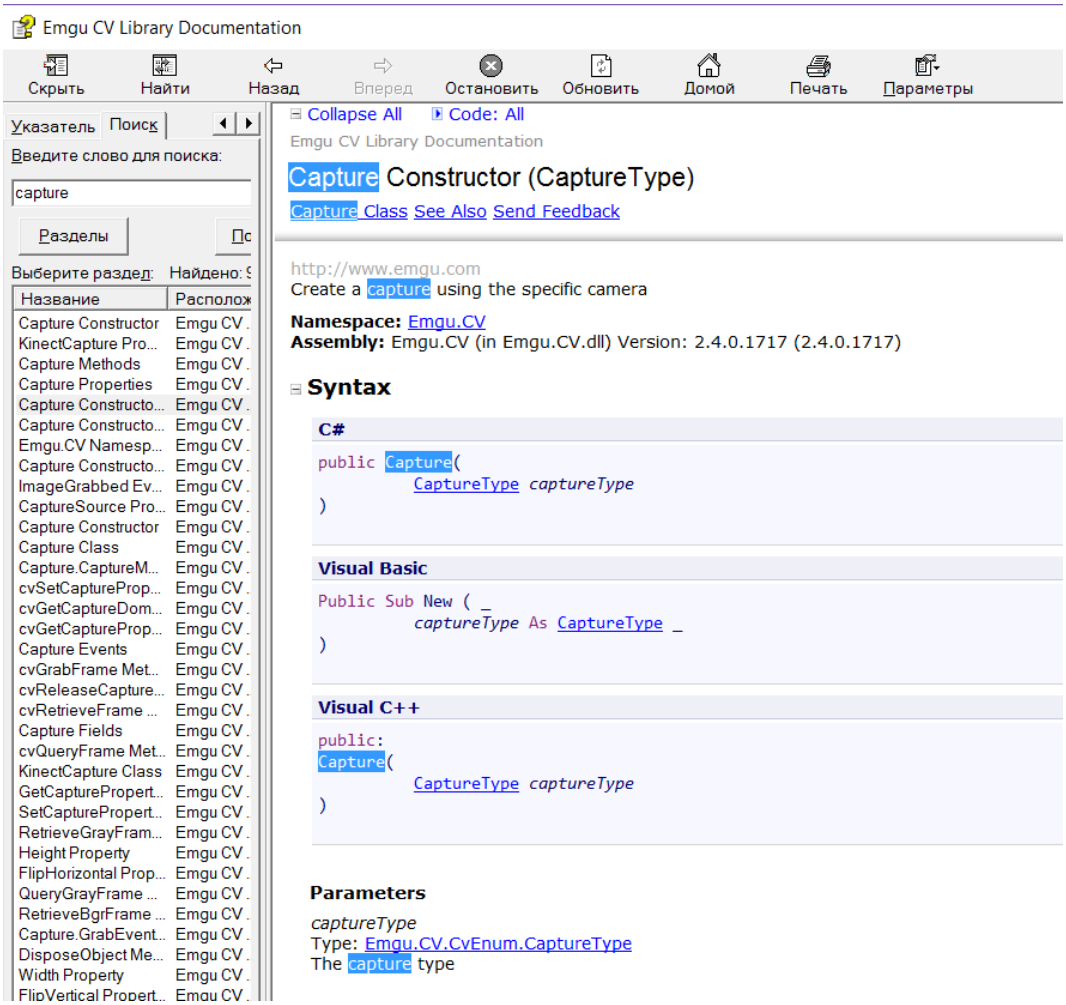
```
Image<Bgr, byte> image = new Image<Bgr, byte>(puth);
Image<Gray, Byte> grayImage = image.Convert<Gray, Byte>();
var Face = grayImage.DetectHaarCascade(faceCascade)[0];
foreach (var face in Face)
{
    image.Draw(face.rect, new Bgr(255, 255, 255), 10);
}
var Eye = grayImage.DetectHaarCascade(eyeCascade)[0];
foreach (var eye in Eye)
{
    image.Draw(eye.rect, new Bgr(0, 0, 255), 3);
}
imageBox1.Image = image;
```

4. Задание

Написать приложение, выполняющее поиск лиц и глаз с изображения, полученного с веб-камеры вашего компьютера, на основе приложения, полученного в ходе данной лабораторной работы.

Подсказка: Для написания данного приложения необходимо использовать класс Capture.

В папке, в которую была распакована библиотека, находится руководство пользования библиотекой на английском языке. Для выполнения данной части лабораторной работы можно использовать данный контент.



Лабораторная работа №2

Разработка приложения с нахождением устойчивых признаков изображения

1. Алгоритм выделения и описания ключевых особенностей изображения SURF

В компьютерном зрении часто используется алгоритм SURF (**Speeded Up Robust Features**) для задач распознавания объекта, регистрации изображения, классификации 3D-реконструкции. До данного алгоритма существовал похожий с ним алгоритм SIFT (**Scale-invariant feature transform [Масштабно-инвариантная функция преобразования]**). Стандартная версия SURF гораздо быстрее SIFT и более устойчива к различным преобразованиям изображения.

Принцип работы алгоритма заключается в поиске особых **ключевых** точек и уникальных **дескрипторов** для них. После этого сравниваются наборы дескрипторов, и выделяется эталонный объект на сцене.

Ключевая точка – точка, которая имеет некие признаки, существенно отличающие ее от основной массы точек (резкие перепады освещенности, углы и т.д.).

Метод ищет ключевые точки с помощью матрицы Гессе. Детерминант матрицы Гессе (гессиан) достигает экстремума в точках максимального изменения градиента яркости. Он хорошо детектирует пятна, углы и края.

Гессиан инвариантен относительно вращения, но не инвариантен масштабу. Поэтому, алгоритм SURF использует разномасштабные фильтры для нахождения гессианов. Используют также разные размеры области, по которой берутся вторые производные.

Для каждой ключевой точки считается направление максимального изменения яркости (градиент) и масштаб, взятый из масштабного коэффициента матрицы Гессе.

Градиент в точке вычисляется при помощи фильтров Хаара.

После выделения ключевых точек SURF формирует их дескрипторы. Вокруг ключевой точки описывается прямоугольная область, которая разбивается на 16 квадрантов одинаковых размеров. Прямоугольная область затем поворачивается в соответствии с ориентацией ключевой точки. На следующем шаге считаются оценки для каждого из 16 квадрантов области при помощи фильтров Хаара.

В результате получается набор из 64 чисел. К описанию точки также добавляется след матрицы Гессе. Вектор и след матрицы вместе образуют **дескриптор** ключевой точки.

Вычисление матрицы Гессе

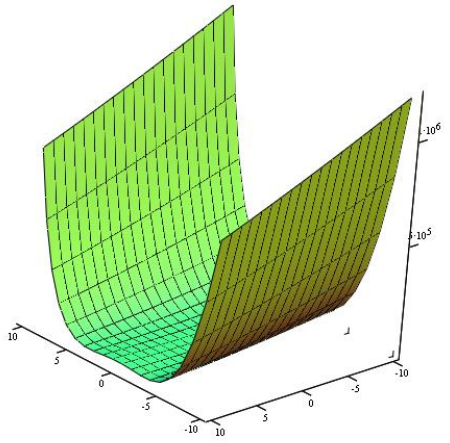
Вычисление экстремума функции (курс методов оптимизации)

Исследуем функцию $\boldsymbol{f(x) = 100(x_1 - x_0^2)^2 + (1 - x_0)^2}$ на нахождение минимума с помощью аналитического метода.

Введем функцию *f(x)* в программу Mathcad:

$$f(x) := 100\left[x_1 - \left(x_0\right)^2\right]^2 + \left(1 - x_0\right)^2$$

Построим график функции:



Найдем градиент функции:

$$\text{grad}(\text{vector}) := \text{return } \nabla_{\text{vector}} f(\text{vector}) \rightarrow \left[400\left[\left(\text{vector}_0\right)^2 - \text{vector}_1\right]\cdot\text{vector}_0 + 2\cdot\text{vector}_0 - 2 \quad 200\,\text{vector}_1 - 200\left(\text{vector}_0\right)^2\right]^T$$

Приравняем градиент функции к нулю и найдем стационарную точку из области действительных чисел:

$$\text{root1} := \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Определим матрицу Гессе:

$$\text{Hesse}(x0, x1) := \begin{bmatrix} \frac{d^2}{dx0^2} f\left(\begin{pmatrix} x0 \\ x1 \end{pmatrix}\right) & \frac{d}{dx0}\left[\frac{d}{dx1} f\left(\begin{pmatrix} x0 \\ x1 \end{pmatrix}\right)\right] \\ \frac{d}{dx1}\left[\frac{d}{dx0} f\left(\begin{pmatrix} x0 \\ x1 \end{pmatrix}\right)\right] & \frac{d^2}{dx1^2} f\left(\begin{pmatrix} x0 \\ x1 \end{pmatrix}\right) \end{bmatrix} \rightarrow \begin{pmatrix} 1200\cdot x0^2 - 400\cdot x1 + 2 & -400\cdot x0 \\ -400\cdot x0 & 200 \end{pmatrix}$$

Найдем матрицу Гессе в точке *root1*:

$$\text{returnHesse}(\text{myX}) := \left|\begin{array}{l} x0 \leftarrow \text{myX}_0 \\ x1 \leftarrow \text{myX}_1 \\ \text{Hesse} \leftarrow \begin{bmatrix} \frac{d^2}{dx0^2} f\left(\begin{pmatrix} x0 \\ x1 \end{pmatrix}\right) & \frac{d}{dx0}\left[\frac{d}{dx1} f\left(\begin{pmatrix} x0 \\ x1 \end{pmatrix}\right)\right] \\ \frac{d}{dx1}\left[\frac{d}{dx0} f\left(\begin{pmatrix} x0 \\ x1 \end{pmatrix}\right)\right] & \frac{d^2}{dx1^2} f\left(\begin{pmatrix} x0 \\ x1 \end{pmatrix}\right) \end{bmatrix} \\ \text{return Hesse} \end{array}\right.$$

$$\text{returnHesse}(\text{root1}) \rightarrow \begin{pmatrix} 802 & -400 \\ -400 & 200 \end{pmatrix}$$

Найдем вектор собственных значений матрицы Гессе:

$$\text{lambda} := \text{eigenvals}(\text{returnHesse}(\text{root1})) \rightarrow \begin{pmatrix} \sqrt{250601} + 501 \\ 501 - \sqrt{250601} \end{pmatrix}$$

Заметим, что оба значения положительны. Это означает, что квадратичная форма и соответствующая ей матрица Гессе положительно определены. Исходя из достаточного условия нахождения экстремума функции, функция $f(x) = 100(x_1 - {x_0}^2)^2 + (1 - x_0)^2$ имеет локальный минимум в точке $\binom{1}{1}$.

```
f(root1) = 0
```

Ответ: Функция $f(x) = 100(x_1 - {x_0}^2)^2 + (1 - x_0)^2$ имеет локальный минимум в точке $\binom{1}{1}$, где принимает значение $f\left(\binom{1}{1}\right) = 0$.

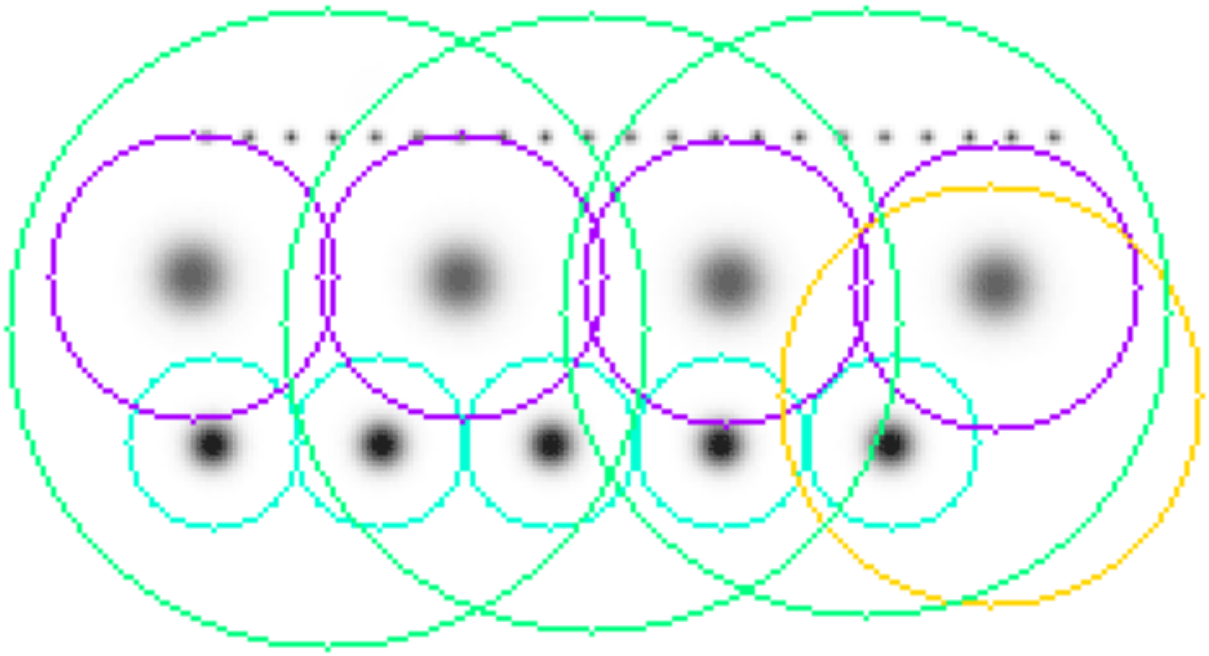
Матрица Гессе для двумерной функции и ее детерминант определяется следующим образом:

$$Hesse\left(f(x,y)\right) = \begin{pmatrix} \frac{\partial^2 f(x,y)}{\partial x^2} & \frac{\partial^2 f(x,y)}{\partial x \partial y} \\ \frac{\partial^2 f(x,y)}{\partial y \partial x} & \frac{\partial^2 f(x,y)}{\partial y^2} \end{pmatrix}$$

Определитель матрицы Гессе:

$$\det(Hesse) = \frac{\partial^2 f(x,y)}{\partial x^2}\frac{\partial^2 f(x,y)}{\partial y^2} - \left(\frac{\partial^2 f(x,y)}{\partial x \partial y}\right)^2$$

Значение гессиана используется для нахождения локального минимума или максимума яркости изображения.



Особые точки представляют собой локальные экстремумы яркости изображения. Мелкие точки не распознаны как особые, из-за порогового отсечения по величине гессиана.
После этих действий SURF накладывает бинаризированные аппроксимации гауссиана или лапласиана и проводит вычисления по определению границ.

Достоинства метода:

1. Инвариантен к поворотам и масштабированию;
2. Инвариантен к разнице общей яркости изображений;
3. Может детектировать более 1 объекта на сцене.

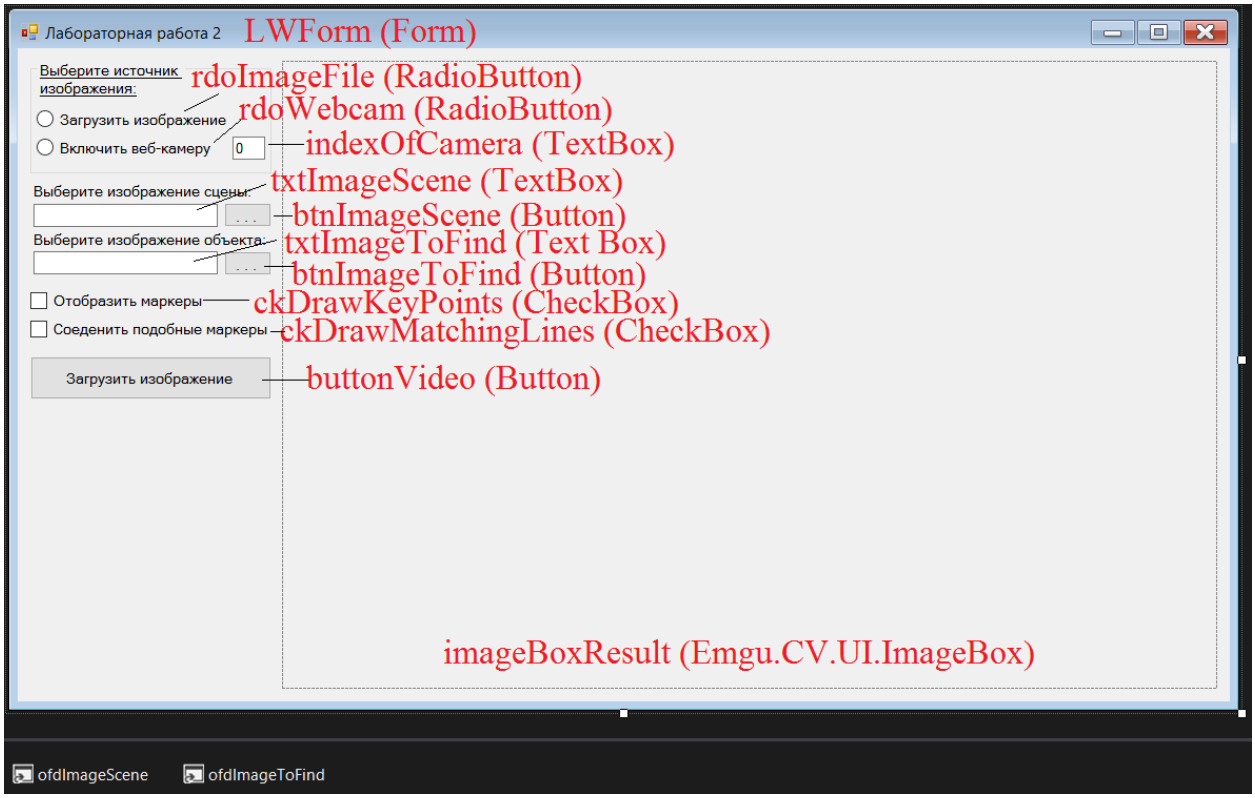
Недостатки метода:

1. Достаточно сложен в реализации;
2. Относительно медленная работа алгоритма (для видео-потока)

2. Разработка приложения. Создание окна для работы с приложением

Для создания главного окна приложения и дальнейшей работы с ним необходимо выполнить шаги 4 – 7 пункта 2 Лабораторной работы 1.

После этого необходимо создать форму главного окна по работе с приложением.



Помимо элементов управления необходимо также добавить диалоговые окна для выбора изображений в интерактивном виде.

3. Разработка приложения. Подключение используемых библиотек

Для работы с приложением необходимо «прописать» несколько библиотек.

```
using Emgu.CV;
using Emgu.CV.CvEnum;
using Emgu.CV.Features2D;
using Emgu.CV.Structure;
using Emgu.CV.UI;
using Emgu.CV.Util;
```

4. Разработка приложения. Создание основных переменных, необходимых для работы

1. Bgr nameVar1 = new Bgr(Color.Blue); – Вместо «Blue» можно использовать другие системные цвета. Переменные данного типа необходимы для выделения ключевых точек, линий, соединяющих изображения, и найденной области образа изображения.
2. SURFDetector nameVar2 = new SURFDetector(500, false); Данный тип переменных используется для работы с алгоритмом SURF. 500 – контрольное значение матрицы Гессе, задающее размер области поиска ключевых точек. Чем больше размер, тем меньше контрольных точек будет выделяться в данной области, что помогает сократить время работы алгоритма, но увеличивает неточность обработки.
3. VectorOfKeyPoint nameVar3 = nameVar2.DetectKeyPointsRaw(nameImageGray, null); Задача данного кода программы заключается в нахождении вектора ключевых точек. nameImageGray (тип Image<Gray, Byte>) – исходное изображение сцены, переведенное в серые тона.
4. Matrix<Single> nameVar4 = nameVar2.ComputeDescriptorsRaw(nameImageGray, null, nameVar3); Нахождение элементов матрицы дескрипторов по вектору ключевых точек.
5. BruteForceMatcher<Single> bruteForceMatcher.KnnMatch(mtxSceneDescriptors, mtxMatchIndices, mtxDistance, intKNumNearestNeighbors, null);

Переменная необходима для перебора всех совпадений. mtxSceneDescriptors (nameVar4) – матрица дескрипторов сцены, mtxMatchIndices – матрица индексов совпадений, mtxDistance – матрица дистанций, intKNumNearestNeighbors – значение количества ближайших соседних точек.

```
private int captureNumber;
private bool blnFirstTimeInResizeEvent = true;
private int intOrigFormWidth, intOrigFormHeight, intOrigImageBoxWidth, intOrigImageBoxHeight;

private Image<Bgr, Byte> imgSceneColor = null;
private Image<Bgr,Byte> imgToFindColor = null;
    private Image<Bgr, Byte> imgCopyOfImageToFindWithBorder = null;

private bool blnImageSceneLoaded = false;
private bool blnImageToFindLoaded = false;

private Image<Bgr, Byte> imgResult = null;

private Bgr bgrKeyPointsColor = new Bgr(Color.Blue);
private Bgr bgrMatchingLinesColor = new Bgr(Color.LightPink);
private Bgr bgrFoundImageColor = new Bgr(Color.Red);

private System.Diagnostics.Stopwatch stopwatch = new System.Diagnostics.Stopwatch();
```

5. Разработка приложения. Создание основной функции работы с приложением

```
private void ProcessFrame(object sender, EventArgs e)
      {
            if (blnImageSceneLoaded == false || blnImageToFindLoaded == false || imgSceneColor == null || imgToFindColor == null)
            {
               this.Text = "Необходимо загрузить изображение";
            }
            this.Text = "Идет загрузка... Пожалуйста, подождите...";
            Application.DoEvents();
            stopwatch.Restart();

          SURFDetector surfDetector = new SURFDetector(500, false);
          Image<Gray, Byte> imgSceneGray = null;
          Image<Gray, Byte> imgToFindGray = null;

          VectorOfKeyPoint vkpSceneKeyPoints, vkpToFindKeyPoints;

          Matrix<Single> mtxSceneDescriptors, mtxToFindDescriptors;

          Matrix<Int32> mtxMatchIndices;
          Matrix<Single> mtxDistance;
          Matrix<Byte> mtxMask;

          BruteForceMatcher<Single> bruteForceMatcher;
          HomographyMatrix homographyMatrix = null;

          int intKNumNearestNeighbors = 2;
          double dblUniquenessThreshold = 0.8;

          int intNumNonZeroElements;

          double dblScaleIncrement = 1.5;
          int intRotationBins = 20;

          double dblRansacReprojectionThreshold = 2.0;

          Rectangle rectImageToFind;
          PointF[] ptfPointsF;
          Point[] ptPoints;

          imgSceneGray = imgSceneColor.Convert<Gray, Byte>();
          imgToFindGray = imgToFindColor.Convert<Gray, Byte>();

          vkpSceneKeyPoints = surfDetector.DetectKeyPointsRaw(imgSceneGray, null);
```

```
        mtxSceneDescriptors = surfDetector.ComputeDescriptorsRaw(imgSceneGray, null, vkpSceneKeyPoints);

        vkpToFindKeyPoints = surfDetector.DetectKeyPointsRaw(imgToFindGray, null);
        mtxToFindDescriptors = surfDetector.ComputeDescriptorsRaw(imgToFindGray, null, vkpToFindKeyPoints);

        bruteForceMatcher = new BruteForceMatcher<Single>(DistanceType.L2);
        bruteForceMatcher.Add(mtxToFindDescriptors);

        mtxMatchIndices = new Matrix<int>(mtxSceneDescriptors.Rows, intKNumNearestNeighbors);
        mtxDistance = new Matrix<Single>(mtxSceneDescriptors.Rows, intKNumNearestNeighbors);

        bruteForceMatcher.KnnMatch(mtxSceneDescriptors, mtxMatchIndices, mtxDistance, intKNumNearestNeighbors, null);

        mtxMask = new Matrix<Byte>(mtxDistance.Rows, 1);
        mtxMask.SetValue(255);

        Features2DToolbox.VoteForUniqueness(mtxDistance, dblUniquenessThreshold, mtxMask);
        intNumNonZeroElements = CvInvoke.cvCountNonZero(mtxMask);

        if(intNumNonZeroElements>=4)
        {
           intNumNonZeroElements = Features2DToolbox.VoteForSizeAndOrientation(vkpToFindKeyPoints, vkpSceneKeyPoints, mtxMatchIndices, mtxMask, dblScaleIncrement, intRotationBins);
           if(intNumNonZeroElements>=4)
           {
              homographyMatrix = Features2DToolbox.GetHomographyMatrixFromMatchedFeatures(vkpToFindKeyPoints, vkpSceneKeyPoints, mtxMatchIndices, mtxMask, dblRansacReprojectionThreshold);
           }
        }

        imgCopyOfImageToFindWithBorder = imgToFindColor.Copy();
        imgCopyOfImageToFindWithBorder.Draw(new Rectangle(1, 1, imgCopyOfImageToFindWithBorder.Width - 3, imgCopyOfImageToFindWithBorder.Height - 3), bgrFoundImageColor, 2);

        if (ckDrawKeyPoints.Checked == true && ckDrawMatchingLines.Checked == true)
```

```
            imgResult = Features2DToolbox.DrawMatches(imgCopyOfImageToFindWithBorder,
                                        vkpToFindKeyPoints,
                                        imgSceneColor,
                                        vkpSceneKeyPoints,
                                        mtxMatchIndices,
                                        bgrMatchingLinesColor,
                                        bgrKeyPointsColor,
                                        mtxMask,
                                        Features2DToolbox.KeypointDrawType.DEFAULT);
        else if (ckDrawKeyPoints.Checked == true && ckDrawMatchingLines.Checked == false)
        {
            imgResult = Features2DToolbox.DrawKeypoints(imgSceneColor, vkpSceneKeyPoints, bgrKeyPointsColor, Features2DToolbox.KeypointDrawType.DEFAULT);
            imgCopyOfImageToFindWithBorder = Features2DToolbox.DrawKeypoints(imgCopyOfImageToFindWithBorder, vkpToFindKeyPoints, bgrKeyPointsColor, Features2DToolbox.KeypointDrawType.DEFAULT);
            imgResult = imgResult.ConcateHorizontal(imgCopyOfImageToFindWithBorder);
        }
        else if (ckDrawKeyPoints.Checked == false && ckDrawMatchingLines.Checked == false)
        {
            imgResult = imgSceneColor;
            imgResult = imgResult.ConcateHorizontal(imgCopyOfImageToFindWithBorder);
        }

        if(homographyMatrix != null)
        {
            rectImageToFind = new Rectangle(0, 0, imgToFindGray.Width, imgToFindGray.Height);

            ptfPointsF = new PointF[]
                        {
                            new PointF(rectImageToFind.Left, rectImageToFind.Top),
                            new PointF(rectImageToFind.Right, rectImageToFind.Top),
                            new PointF(rectImageToFind.Right, rectImageToFind.Bottom),
                            new PointF(rectImageToFind.Left, rectImageToFind.Bottom)
                        };

            homographyMatrix.ProjectPoints(ptfPointsF);
            ptPoints = new Point[]
                        {
```

```
                                Point.Round(ptfPointsF[0]),
                                Point.Round(ptfPointsF[1]),
                                Point.Round(ptfPointsF[2]),
                                Point.Round(ptfPointsF[3]),
                            };

                imgResult.DrawPolyline(ptPoints, true, bgrFoundImageColor, 2);
            }
            imageBoxResult.Image = imgResult;
            if(rdoImageFile.Checked)
            {
                stopwatch.Stop();
                this.Text = "Процессорное время = " + stopwatch.Elapsed.TotalSeconds.ToString();
            }
        }
```

6. Разработка приложения. Создание обработчика события изменения размера экрана

```
private void LWForm_Resize(object sender, EventArgs e)
        {
            if (blnFirstTimeInResizeEvent)
                blnFirstTimeInResizeEvent = false;
            else
            {
                imageBoxResult.Width = this.Width - (intOrigFormWidth - intOrigImageBoxWidth);
                imageBoxResult.Height = this.Height - (intOrigImageBoxHeight - intOrigImageBoxHeight);
            }
        }
```

7. Разработка приложения. Создание обработчика события нажатия на кнопку buttonVideo

```
private void buttonVideo_Click(object sender, EventArgs e)
        {
        if (txtImageToFind.Text != String.Empty && txtImageScene.Text != String.Empty)
                ProcessFrame(new Object(), new EventArgs());
        else
                this.Text = "Выберите изображение";
        }
```

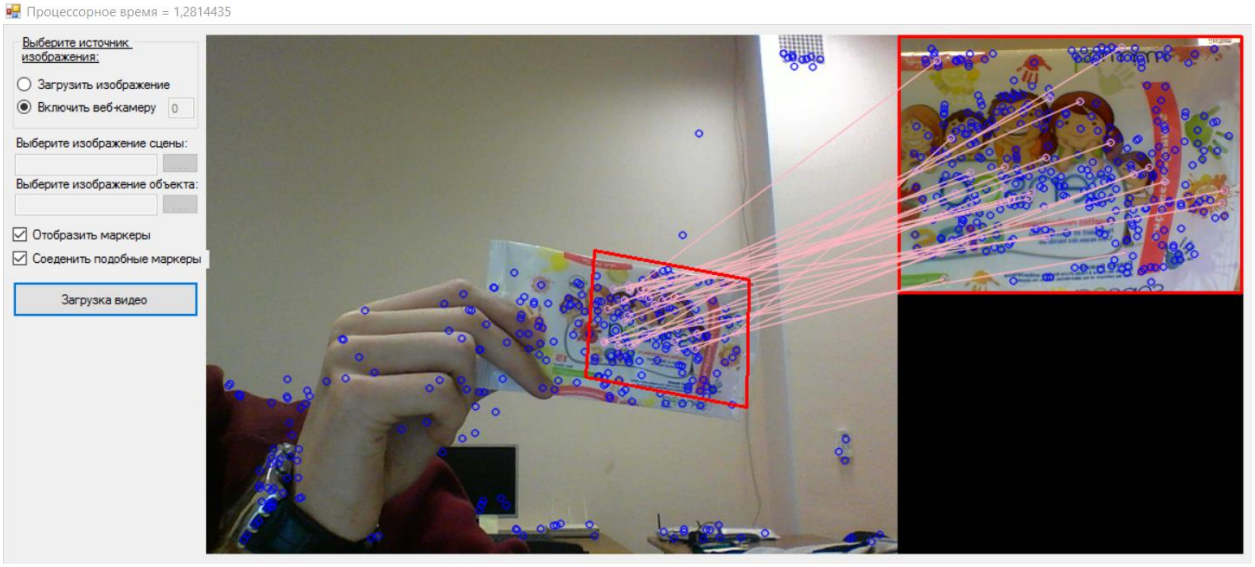
8. Результат работы программы

В результате работы программы мы должны получить исходное изображение сцены с выделенной областью загруженного образа.

9. Задание

В качестве задания для данной лабораторной работы необходимо разработать приложение, работающее с изображением, полученным с камеры Вашего устройства.

В результате работы должен происходить поиск образа изображения в реальном времени.



Лабораторная работа №3

Разработка приложения с использованием контурного анализа

1. Основы контурного анализа

Контурный анализ – анализ выделенной области в компьютерном зрении, который позволяет описывать, хранить, сравнивать и находить объекты, представленные в виде внешних очертаний – контуров.

Предполагается, что контур содержит всю необходимую информацию о форме объекта.

Контурный анализ позволяет эффективно решать основные проблемы распознавания образов – переноса, поворота и масштабирования изображения объекта.

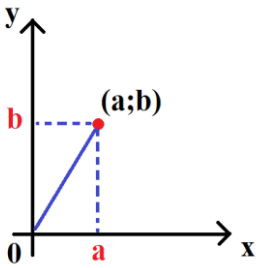
Контур – граница объекта, множество точек (пикселей), отделенных от фона изображения.

В большинстве случаев для кодирования контуров используются различные форматы, связанные с матрицами координат точек (двумерное кодирование, код Фримена).

Контурный анализ и не использует данные матрицы. Вместо них используются последовательности комплексных чисел.

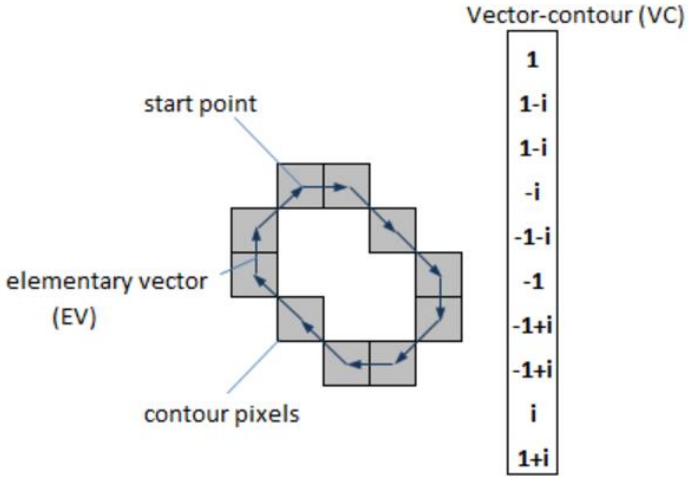
2. Алгоритм получения координат точек в виде последовательности комплексных чисел.

Любую точку можно представить в виде комплексного числа. Например, в данном случае точка будет иметь вид ***(a+ib)***.



Для получения последовательности комплексных чисел заданного контура необходимо выполнить следующий алгоритм:

1. Выбрать отправную (стартовую) точку.



2. Выбрать направление обхода (по часовой стрелке или против часовой стрелки).
3. Относительно начальной точки необходимо перемещаться по контуру в заданном направлении.
    a. В случае смещения пикселя контура вправо (влево) вектор будет иметь вид «+1» («–1»)

b. В случае смещения пикселя контура вверх (вниз) вектор будет иметь вид «+i» («–i»).

Каждый вектор контура будет называться элементарным вектором (EV), а последовательность элементарных векторов – вектор-контур (VC).

Таким образом, вектор-контур длины k будет выглядеть следующим образом:

$$\Gamma = (ev_0, ev_1, \dots, ev_{k-1})$$

Таким образом, используя координаты векторов в комплексном виде, мы можем использовать упрощенное (по отношению к скалярному произведению векторов, записанных в виде двумерных координат на плоскости) скалярное произведение комплексных чисел.

3. Свойства контуров

1. Алгебраическая сумма элементарных векторов замкнутого контура равна нулю.
2. Контур-вектор не зависит от параллельного переноса исходного изображения.
3. При повороте изображения на определенный угол, каждый элементарный вектор контура будет эквивалентно повернут на тот же угол.
4. При изменении стартовой точки, вектор и его дальнейшее использование меняться не будет, поскольку элементарные вектора кодируются относительно друг друга. Не имеет значения, какой элементарный вектор будет стоять на первом месте.
5. Масштабирование исходного изображения сводится к умножению каждого элементарного вектора контура на коэффициент масштабирования.

4. Скалярное произведение векторов

Скалярное произведение векторов, обозначающих контуры Г и К:

$$(\Gamma, К) = \sum_{i=0}^{k-1} (ev\Gamma_i, evК_i)$$

Скалярное произведение скалярных чисел:

$$(a + ib, c + id) = (a + ib)(c - id) = ac + bd + i(bc - ad)$$

Скалярное произведение обычных векторов:

$$\big((a; b), (c; d)\big) = ac + bd$$

Результат скалярного произведения векторов является действительным числом. Результат произведения комплексных чисел – комплексное число.

Действительная часть скалярного произведения комплексных чисел совпадает со скалярным произведением соответствующих векторов.

Скалярное произведение векторов – произведение длин векторов на косинус угла между ними.

Нормализованное скалярное произведение (NSP)

$$NSP = \frac{(\Gamma, К)}{|\Gamma||К|}$$

Норма вектора:

$$|\Gamma| = \sqrt{\sum_{i=0}^{k-1} |ev_i|^2}$$

NSP в пространстве комплексных чисел также является комплексным числом.

Наибольшее возможное значение нормы вектора:

$$\Gamma = u\text{К}$$

u – произвольное комплексное число.

Таким образом, NSP контура Г достигнет наибольшего значения только в том случае, если контур К будет тем же контуром, но масштабированным или повернутым на какой-то угол. Норма вектора-контура инвариантна к изменению масштаба, параллельного переноса и поворота.

Норма вектора-контура является мерой близости контура.

Норма вектора-контура зависит от выбора стартовой точки.

5. Корреляционные функции в контурном анализе

Интеркорреляционная функция (ICF).

$$t(m) = \left(\Gamma, \text{К}^{(m)}\right), m = 0,1, \dots, k-1$$

$\text{К}^{(m)}$ – контур, полученный из контура сдвигами цикла его элементарных векторов относительно начальной точки.

ICF с учетом максимальной нормы:

$$t_{max} = \max\left(\frac{t(m)}{|\Gamma||\text{К}|}\right), , m = 0,1, \dots, k-1$$

Учитывая это, можно сказать, что t_{max} является мерой подобия двух контуров, инвариантных к параллельному переносу, масштабированию и вращению.

Автокорреляционная функция (ACF) – интеркорреляционная функция, в которой контур Г равен контуру К при различных сдвигах начальной точки.

$$v(m) = \left(\Gamma, \Gamma^{(m)}\right), m = 0,1, \dots, k-1$$

- ACF не зависит от выбора начальной точки.
- ACF является симметричной относительно середины вектора.

6. Общий алгоритм распознавания

1. Предварительная обработка изображения (сглаживание, фильтрация шума, увеличение контраста)
2. Бинаризация изображения и выбор контуров объекта
3. Первичная фильтрация контуров
4. Уравнивание контуров (усреднение длины), сглаживание
5. Поиск всех обнаруженных контуров, поиск максимально аналогичного шаблона заданному контуру.

7. Уравнивание контуров

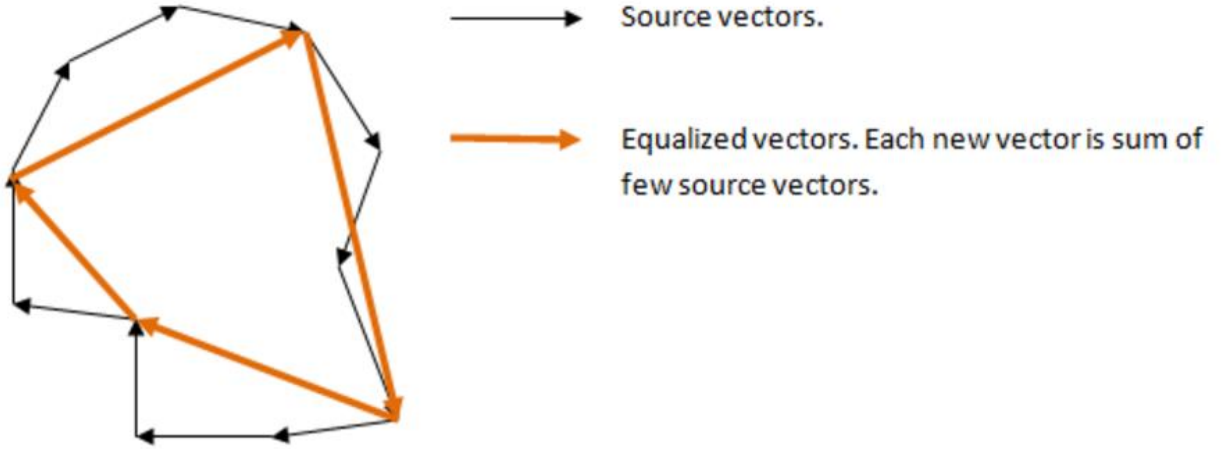
Методы контурного анализа используют вычисления, которые можно рассчитывать только в случае двух контуров, вектора-контуры которых должны иметь одинаковую длину. В контурах реальных изображений это встречается крайне редко.

Для поиска и сравнения контуров они все должны быть приведены к одной длине. Данный процесс называется уравниванием.

1. Сначала необходимо зафиксировать длину вектор-контура, который мы будем использовать в качестве образа. Обозначим ее *k*.
2. Для каждого исходного контура А мы создаем вектор-контур Г, длина которого будет равна *k*.

3. Если необходимый контур имеет больше элементарных векторов, чем число *k*, тогда необходимо выбрать только то количество точек, которое нужно для выделения контура размерности *k*.

```
Complex[] newPoint = new Complex[newCount];

for (int i = 0; i < Count; i++)
    newPoint[i * newCount / Count] += this[i];
```



4. Если необходимый контур имеет меньшее количество элементарных векторов, чем *k*, тогда необходимо интерполировать данный контур.

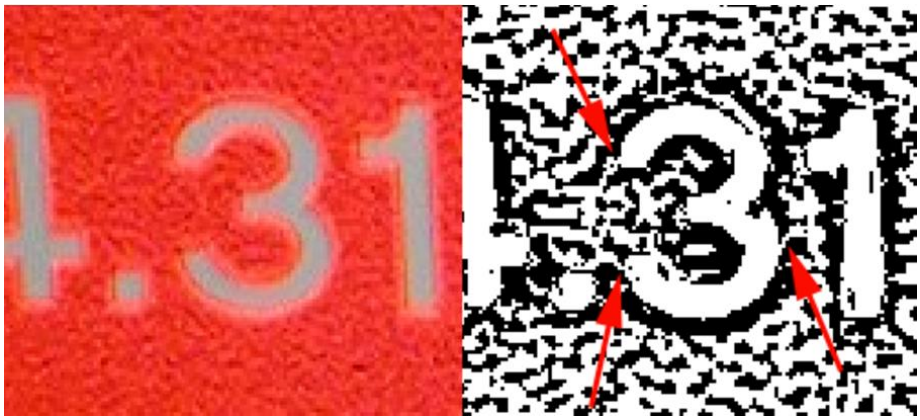
```
Complex[] newPoint = new Complex[newCount];

for (int i = 0; i < newCount; i++)
{
    double index = 1d * i * Count / newCount;
    int j = (int)index;
    double k = index - j;
    newPoint[i] = this[j] * (1 - k) + this[j + 1] * k;
}
```

Остается вопрос, какое значение количества элементарных векторов является оптимальным? Ответ на этот вопрос полностью определяется спецификой области применения данного алгоритма. С одной стороны, большое количество EV затрудняет расчеты, увеличивая большие расходы в оценке эффективности алгоритма, с другой – малое количество EV имеет меньше информации, точность вычислений ухудшается, а уровень распознавания шума – увеличивается.

8. Ограничения контурного анализа

Контурный анализ имеет две отрицательные черты, влияющие на результаты распознавания.

- Проблема выбора контура в изображениях. Контур – это строго определенная дискретная структура. Реальный объект не может иметь четкую границу, он может сливаться с фоном (цвет объекта совпадает с цветом фона).



- Методы контурного анализа описывают объекты в отдельности, и не допускают пересечений с другими объектами

Таким образом, контурный анализ имеет слабую устойчивость к помехам, не допускает пересечений и частичных видимостей объектов.

9. Библиотека контурного анализа ContourAnalysis

Класс *Countour* необходим для создания и сохранения контуров. Он содержит основные операции для контуров – скалярное произведение, масштабирование, выравнивание, нормализацию, оценку спектра, вычисление интеркорреляционной (ICF) и автокорреляционной (ACF) функций и т.д.

Класс *Template* используется для создания базы шаблонов. Этот класс хранит контур, его ACF и дескрипторы ACF. Так же, в нем хранятся линейные параметры начального контура.

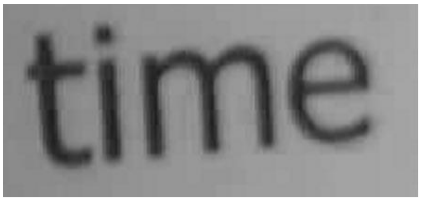
Класс *TemplateFinder* реализует быстрый поиск шаблона для полученного контура. Результат работы этого класса *FoundTemplateDesc*, который содержит начальный контур и найденный шаблон для данного контура. Также, в нем содержится значение коэффициента сходства, угол поворота и масштаб контура, относительно шаблона.

Класс *ImageProcessor* используется для обработки изображений. Он также хранит базу шаблонов.

Метод *ImageProcessor.ProcessImage()* на входе принимает изображение. Результатом его работы является список обнаруженных контуров *ImageProcessor.samples* и перечень признанных контуров *ImageProcessor.foundTemplates*.

Алгоритм работы *ImageProcessor*

1. Изображение преобразуется в оттенках серого цвета

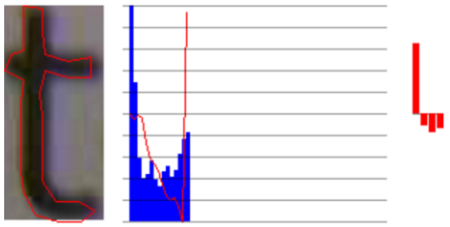


2. Затем изображение переводится в бинарный вид методом *AdaptiveThereshold*

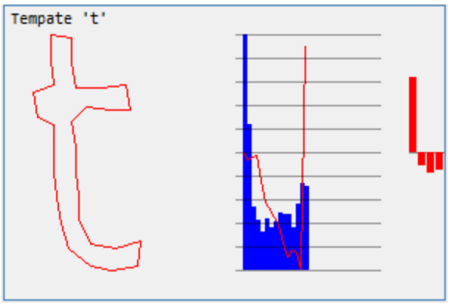
3. После этого извлекаются контуры

4. Происходит фильтрация по линейным параметрам (длина, площадь и т.д.)

5. Контуры уравниваются, вычисляются ACF и дескрипторы для ACF



6. Создается максимально подходящий шаблон



Статический класс *TemplateGenerator* используется для автоматической генерации шаблонов букв и цифр по загруженному шрифту.

10. Задание

Используя полученные от преподавателя файлы с классами (с расширением *.cs), написать приложение по распознаванию текста по вариантам.