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

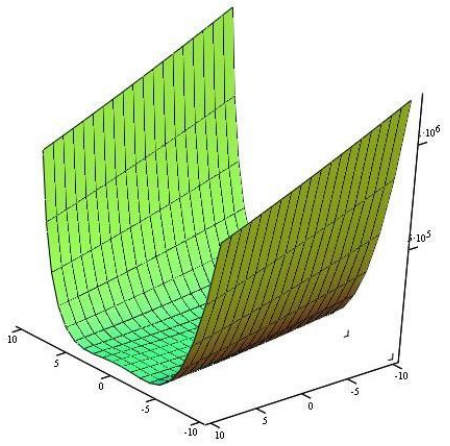
Вычисление экстремума функции (курс методов оптимизации)

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:

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

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 $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

f(root1) = 0

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

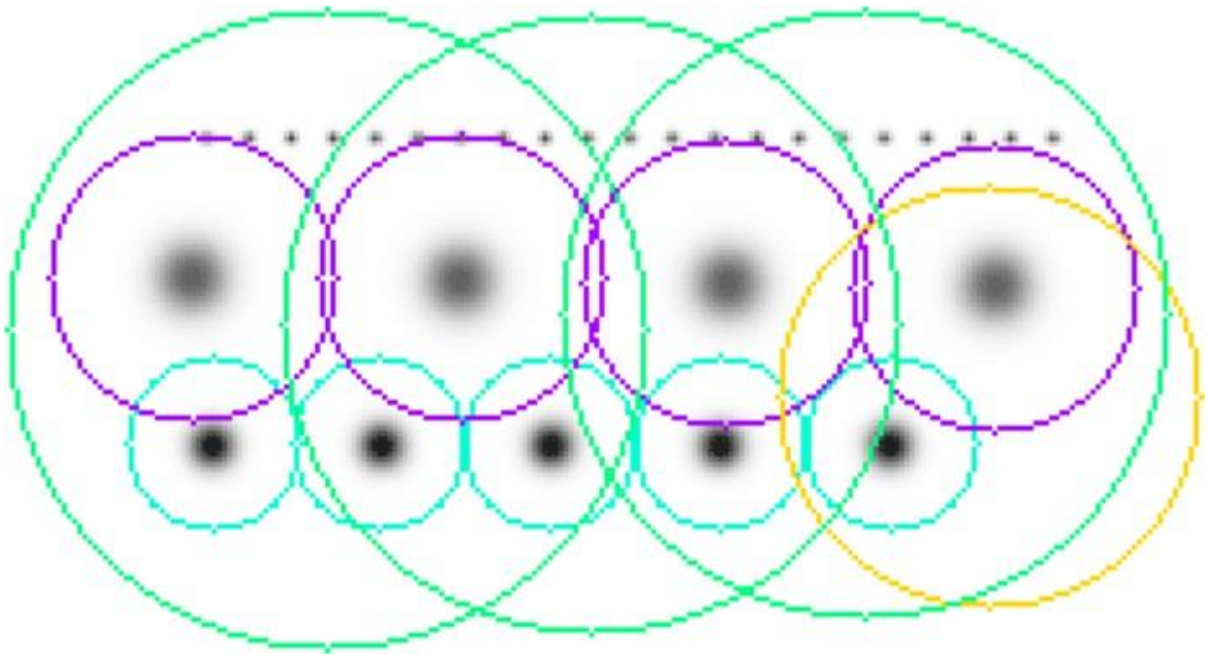
Матрица Гессе для двумерной функции и ее детерминант определяется следующим образом:

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

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.

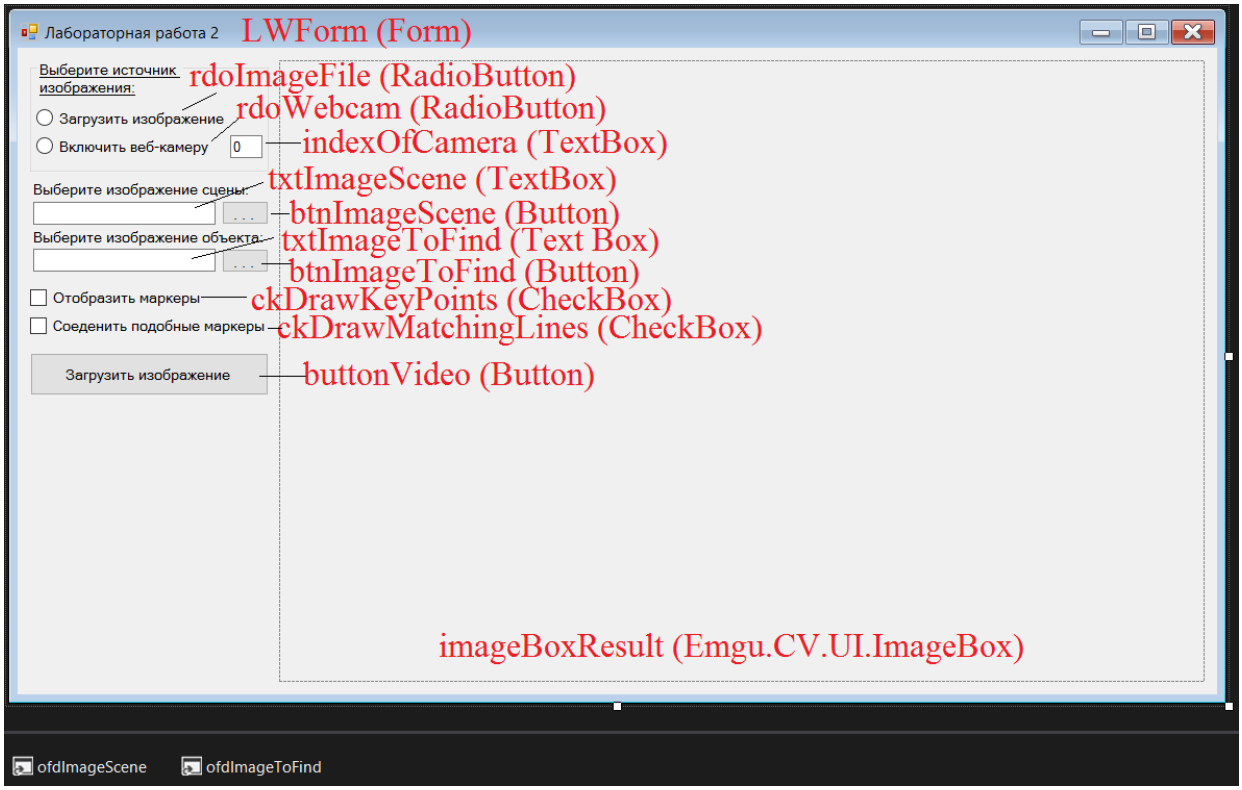
Недостатки метода:

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

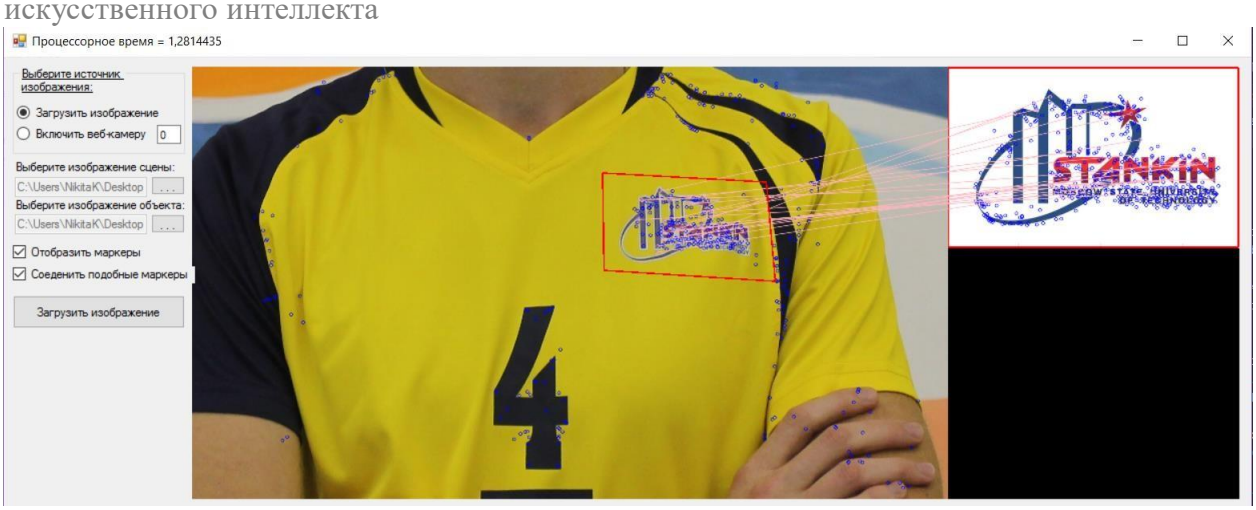
```
private void LWForm_Resize(object sender, EventArgs e)
    {
        if (blnFirstTimeInResizeEvent)
            blnFirstTimeInResizeEvent = false;
        else
        {
            imageBoxResult.Width = this.Width - (intOrigFormWidth - intOrigImageBoxWidth);
            imageBoxResult.Height =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