Название работы: Стабилизация изображения и поиск соответствий

Цель работы: Целью данной работы является изучение методик определения смещения характерных точек на изображении, а также поиска соответствий характерных точек на разных изображениях.

Постановка задачи:

Необходимо разработать приложение Windows Forms, способное осуществлять:

1. Поиск характерных точек на изображении.
2. Поиск смещения характерных точек при изменении изображения.
3. Поиск соответствия характерных точек разных изображений.

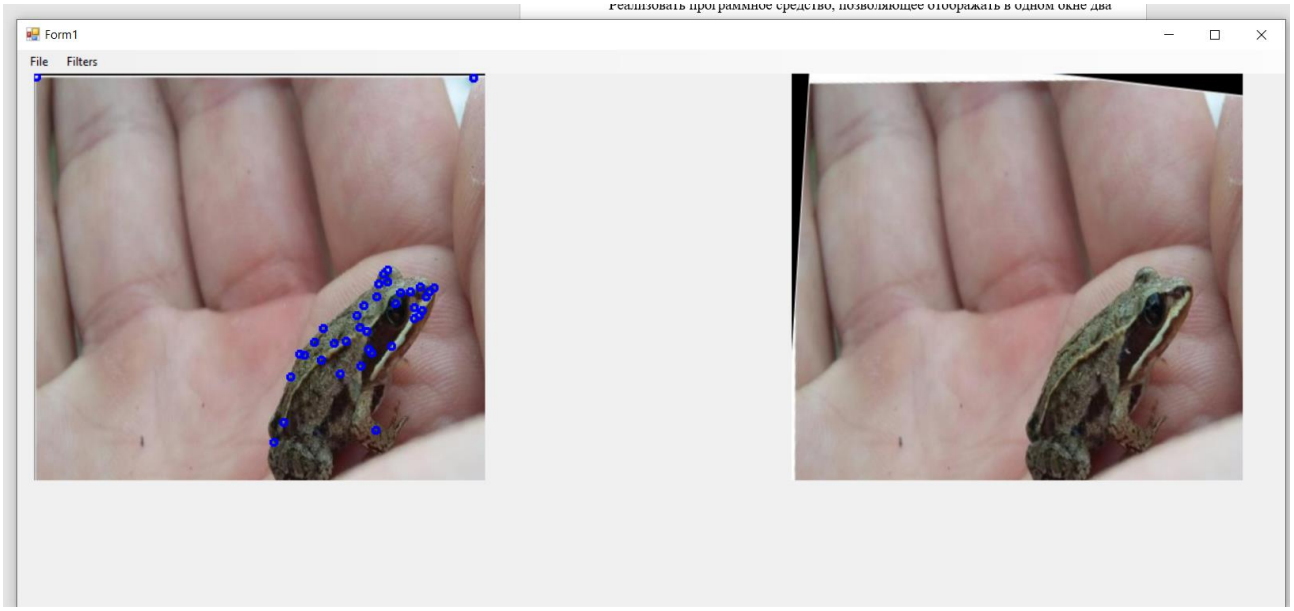
Задание:

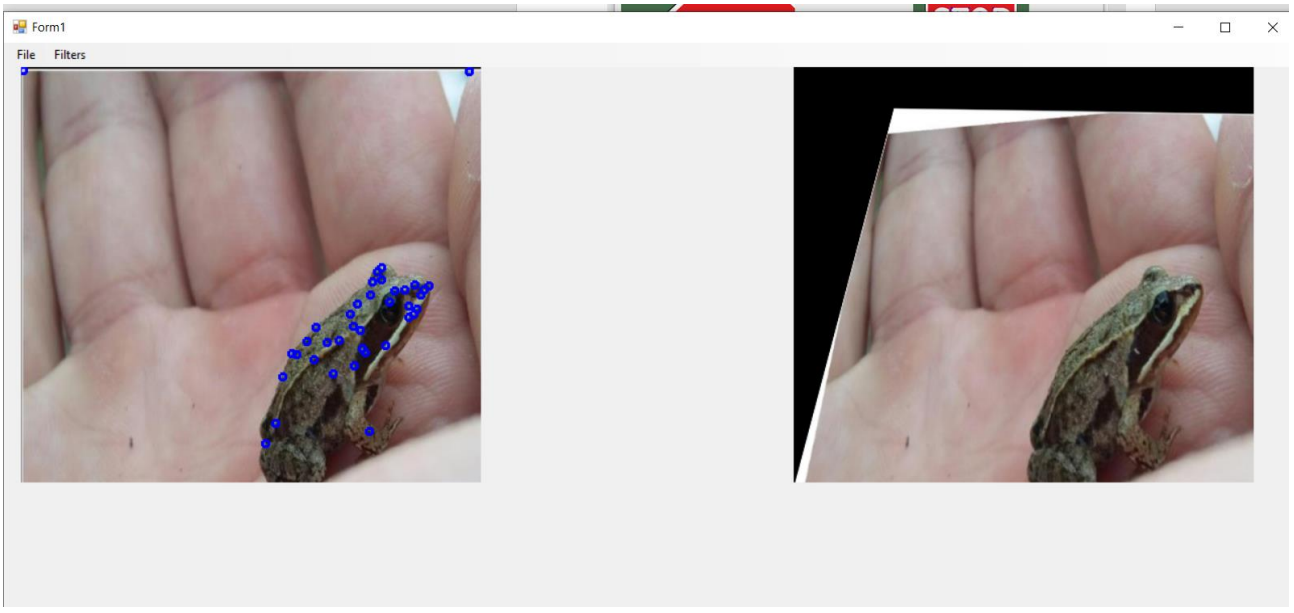
Реализовать программное средство, позволяющее отображать в одном окне два изображения, “оригинальное” слева и “результат обработки” справа.

Реализовать интерфейс, позволяющий по нажатию на соответствующие кнопки выполнять следующие операции:

1. Вычисление изменений позиций характерных точек при помощи метода Лукаса-Канаде.
2. Поиск общих характерных точек изображений при помощи сравнения характерных точек

Реализовать программное средство, позволяющее отображать в одном окне два





Листинг кода:

```
using Emgu.CV.CvEnum;
using Emgu.CV.Structure;
using Emgu.CV;
using Emgu.CV.Features2D;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Emgu.CV.XObjdetect;
using Emgu.CV.Util;

namespace Roviac_7
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        public Image<Bgr, byte> sourceImage { get; set; }
        public Image<Bgr, byte> twistedImage { get; set; }
        private void openToolStripMenuItem_Click(object sender, EventArgs e)
```

```
        {
            OpenFileDialog ofd = new OpenFileDialog();
            ofd.Filter = "Picture Files (*jpg, *.png)| *.jpg;*.png";
            var result = ofd.ShowDialog();

            if (result == DialogResult.OK)
            {
                string filename = ofd.FileName;
                sourceImage = new Image<Bgr, byte>(filename);
                sourceImage = sourceImage.Resize(608, 506, Inter.Linear);
                imageBox1.Image = sourceImage.Resize(608, 506, Inter.Linear);
            }
        }

        public static Image<Bgr, byte> Characteristic_points(Image<Bgr, byte>
sourceImage)
        {
            GFTTDetector detector = new GFTTDetector(40, 0.01, 5, 3, true);
            MKeyPoint[] GFP1 = detector.Detect(sourceImage.Convert<Gray,
byte>().Mat);


            var output = sourceImage.Copy();
            foreach (MKeyPoint p in GFP1)
            {
                CvInvoke.Circle(output, Point.Round(p.Point), 3, new
Bgr(Color.Blue).MCvScalar, 2);
            }

            return output.Resize(608, 506, Inter.Linear) ;
```

```
        }

        private void обнаружениеХарактерныхТочекToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            imageBox2.Image = Characteristic_points(sourceImage);
        }

        private void gFTTДетекторToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            imageBox2.Image = Characteristic_points(sourceImage);
        }

        public static Image<Bgr, byte> Brisk_detector(Image<Bgr, byte> sourceImage)
        {
            Brisk detector = new Brisk();

            MKeyPoint[] GFP1 = detector.Detect(sourceImage.Convert<Gray,
byte>().Mat);


            var output = sourceImage.Copy();
            foreach (MKeyPoint p in GFP1)
            {
                CvInvoke.Circle(output, Point.Round(p.Point), 3, new
Bgr(Color.Blue).MCvScalar, 2);
            }

            return output.Resize(608, 506, Inter.Linear);
```

```
        }
        private void bRISKДетекторToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            imageBox2.Image = Brisk_detector(sourceImage);
        }


        public static Image<Bgr, byte> Fast_detector(Image<Bgr, byte> sourceImage)
        {
            FastFeatureDetector detector = new FastFeatureDetector();

            MKeyPoint[] GFP1 = detector.Detect(sourceImage.Convert<Gray,
byte>().Mat);


            var output = sourceImage.Copy();
            foreach (MKeyPoint p in GFP1)
            {
                CvInvoke.Circle(output, Point.Round(p.Point), 3, new
Bgr(Color.Blue).MCvScalar, 2);
            }

            return output.Resize(608, 506, Inter.Linear);
        }
        private void fASTДетекторToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            imageBox2.Image = Fast_detector(sourceImage);
        }
```

```
        private void openTwistedImgToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            OpenFileDialog ofd = new OpenFileDialog();
            ofd.Filter = "Picture Files (*jpg, *.png)| *.jpg;*.png";
            var result = ofd.ShowDialog();

            if (result == DialogResult.OK)
            {
                string filename = ofd.FileName;
                twistedImage = new Image<Bgr, byte>(filename);
                twistedImage = twistedImage.Resize(608, 506, Inter.Linear);
                imageBox2.Image = twistedImage.Resize(608, 506, Inter.Linear);
            }
        }

        //FastFeatureDetector detector = new FastFeatureDetector();
        public void Optical_flow(Image<Bgr, byte> baseImg, Image<Bgr, byte>
twistedImg)
        {
            GFTTDetector detector = new GFTTDetector(40, 0.01, 5, 3, true);
            MKeyPoint[] GFP1 = detector.Detect(baseImg.Convert<Gray, byte>().Mat);

            //MKeyPoint[] GFP2 = detector.Detect(twistedImg.Convert<Gray,
byte>().Mat);

            //создание массива характерных точек исходного изображения (только
позиции)
            PointF[] srcPoints = new PointF[GFP1.Length];
```

```
for (int i = 0; i < GFP1.Length; i++)
   srcPoints[i] = GFP1[i].Point;

PointF[] destPoints; //массив для хранения позиций точек на изменённом изображении

//PointF[] destPoints = new PointF[GFP2.Length]; //массив для хранения позиций точек на изменённом изображении
//for (int i = 0; i < GFP2.Length; i++)
//    destPoints[i] = GFP2[i].Point;

imageBox1.Image = Characteristic_points(baseImg);
imageBox2.Image = Characteristic_points(twistedImg);

byte[] status; //статус точек (найдены/не найдены)
float[] trackErrors; //ошибки
                    //вычисление позиций характерных точек на новом изображении методом Лукаса-Канаде
CvInvoke.CalcOpticalFlowPyrLK(
 baseImg.Convert<Gray, byte>().Mat, //исходное изображение
 twistedImg.Convert<Gray, byte>().Mat,//изменённое изображение
 srcPoints, //массив характерных точек исходного изображения
 new Size(20, 20), //размер окна поиска
 5, //уровни пирамиды
 new MCvTermCriteria(20, 1), //условие остановки вычисления оптического потока
 out destPoints, //позиции характерных точек на новом изображении
 out status, //содержит 1 в элементах, для которых поток был найден
 out trackErrors //содержит ошибки
 );
```

```
            //вычисление матрицы гомографии
            Mat homographyMatrix = CvInvoke.FindHomography(destPoints, srcPoints,
            RobustEstimationAlgorithm.LMEDS);


            var destImage = new Image<Bgr, byte>(baseImg.Size);
            CvInvoke.WarpPerspective(twistedImg, destImage, homographyMatrix,
destImage.Size);

            imageBox2.Image = destImage.Resize(608, 506, Inter.Linear);
        }

        private void вычислениеОптическогоПотокаToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            Optical_flow(sourceImage, twistedImage);
        }

        public static Image<Bgr, byte> Feature_point_comparison(Image<Bgr, byte>
sourceImage, Image<Bgr, byte> twistedImage)
        {

            GFTTDetector detector = new GFTTDetector(40, 0.01, 5, 3, true);
            //MKeyPoint[] GFP1 = detector.Detect(baseImg.Convert<Gray, byte>().Mat);

            var twistedImgGray = twistedImage.Convert<Gray, byte>();
            var baseImgGray = sourceImage.Convert<Gray, byte>();
```

```
//генератор описания ключевых точек
Brisk descriptor = new Brisk();
//поскольку в данном случае необходимо посчитать обратное преобразование
//базой будет являться изменённое изображение
VectorOfKeyPoint GFP1 = new VectorOfKeyPoint();
UMat baseDesc = new UMat();
UMat bimg = twistedImgGray.Mat.GetUMat(AccessType.Read);
VectorOfKeyPoint GFP2 = new VectorOfKeyPoint();
UMat twistedDesc = new UMat();
UMat timg = baseImgGray.Mat.GetUMat(AccessType.Read);
//получение необработанной информации о характерных точках изображений
detector.DetectRaw(bimg, GFP1);
//генерация описания характерных точек изображений
descriptor.Compute(bimg, GFP1, baseDesc);
detector.DetectRaw(timg, GFP2);
descriptor.Compute(timg, GFP2, twistedDesc);

//класс позволяющий сравнивать описания наборов ключевых точек
BFMatcher matcher = new BFMatcher(DistanceType.L2);

//массив для хранения совпадений характерных точек
VectorOfVectorOfDMatch matches = new VectorOfVectorOfDMatch();
//добавление описания базовых точек
matcher.Add(baseDesc);
//сравнение с описанием изменённых
matcher.KnnMatch(twistedDesc, matches, 2, null);
//3й параметр - количество ближайших соседей среди которых осуществляется поиск совпадений
```

```
//4й параметр - маска, в данном случае не нужна

//маска для определения отбрасываемых значений (аномальных и не уникальных)
Mat mask = new Mat(matches.Size, 1, DepthType.Cv8U, 1);
mask.SetTo(new MCvScalar(255));
//определение уникальных совпадений
Features2DToolbox.VoteForUniqueness(matches, 0.8, mask);

//отбрасывание совпадения, чьи параметры масштабирования и поворота не совпадают с параметрами большинства
int nonZeroCount = Features2DToolbox.VoteForSizeAndOrientation(GFP1, GFP1, matches, mask, 1.5, 20);

var res = sourceImage.Clone();

Features2DToolbox.DrawMatches(twistedImage, GFP1, sourceImage, GFP2, matches, res, new MCvScalar(255, 0, 0), new MCvScalar(255, 0, 0), mask);

Mat homographyMatrix;
//получение матрицы гомографии
homographyMatrix = Features2DToolbox.GetHomographyMatrixFromMatchedFeatures(GFP1, GFP2, matches, mask, 2);

var destImage = new Image<Bgr, byte>(sourceImage.Size);
CvInvoke.WarpPerspective(twistedImage, destImage, homographyMatrix, destImage.Size);

return destImage.Resize(608, 506, Inter.Linear);
```

```
        }

        private void сравнениеХарактерныхТочекToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            imageBox2.Image = Feature_point_comparison(sourceImage, twistedImage);
        }
    }
}
```