Лабораторная работа №7:
Стабилизация изображения и поиск соответствий

Цель:

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

План работ:

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.

Обнаружение характерных точек:

Предполагается, что каждый объект на изображении имеет определённые точки (области пикселей), отличающиеся от остальных по яркостным или пространственным характеристикам. Эти точки называются характерными, или точками интереса. Обнаружив подобные точки для изображения или его отдельных сегментов, можно установить соответствие межу фрагментами разных изображений, установить наличие общих объектов или произошедшие изменения.

Подключение пространства имён, содержащее детекторы характерных точек:

```
using Emgu.CV.Features2D;
```

Создание объекта, реализующего Good Features To Track детектор характерных точек:

```
GFTTDetector detector = new GFTTDetector(40, 0.01, 5, 3, true);
```

где:

первый параметр – число характерных точек, возвращаемое детектором

второй параметр – минимально допустимое качество находимых характерных точек

третий параметр – минимальное расстояние между найденными характерными точками

четвёртый параметр – размер области, внутри которой осуществляет поиск характерной точки

пятый параметр – используется ли детектор углов Харриса

Альтернативный вариант, использование BRISK детектора:

```
Brisk detector = new Brisk();
```

Или FAST детектора:

```
FastDetector detector = new FastDetector();
```

Вызов метода определения характерных точек:

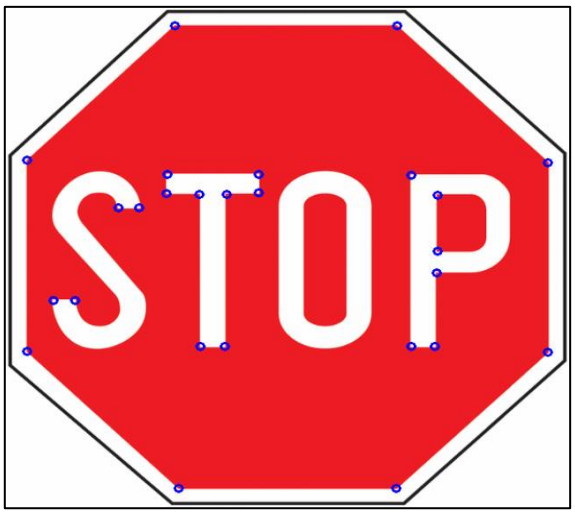
```
MKeyPoint[] GFP1 = detector.Detect(baseImg.Convert<Gray, byte>().Mat);
```

Отобразить области содержащей характерные точки можно при помощи окружностей:

```
foreach (MKeyPoint p in modelKeyPoints)
{
    CvInvoke.Circle(output, Point.Round(p.Point), 3, new Bgr(Color.Blue).MCvScalar, 2);
}
```

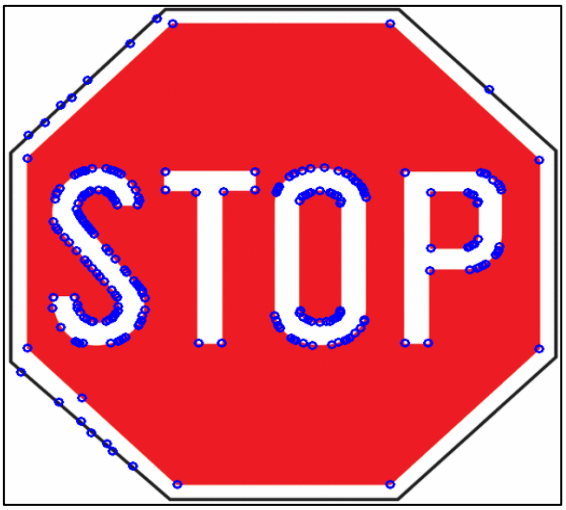
В библиотеке OpenCV содержится реализация многих детекторов характерных точек, но для выполнения работы достаточно этих трёх. Основным детектором будет являться GFTT.

Результат работы GFTT детектора:



Результат работы BRISK детектора:

Результат работы FAST детектора:



Как видно из примера, GFTT с включённым детектором Харриса записывает в характерные точки в основном углы, BRISK детектор записывает искривлённые поверхности, а FAST находит много “лишних” точек.

Вычисление оптического потока.

Предположим, что на вход программы поступает последовательность изображений, отличающихся на небольшое смещение или поворот (например, видеопоток).

Пример, слева оригинальное изображение (baseImg), справа изменённое (twistedImg):

Необходимо вычислить позиции характерных точек исходного (левого изображений) на изменённом (правом). Одним из способов реализовать это является вычисление оптического потока изображения методом Лукаса-Канаде.

В библиотеке EmguCV вызов метода Лукаса-Канаде реализован следующим образом:

```
//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

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
    );
```

Применив этот метод к исходному изображению, изменённому изображению и характерным точкам исходного изображения, можно получить позиции характерных точек на изменённом изображении.

Пример:

Как было описано в лабораторной работе номер три, имея два набора точек, можно найти гомографическую проекцию одного набора на другой, то есть, вычислить обратное преобразование, позволяющее вернуть изменённое изображение в границы исходного.

Сделать это можно при помощи функции FindHomography:

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

Где `HomographyMethod.LMEDS` – метод гомографии.

После чего, достаточно применить полученное преобразование к изменённому изображению:

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

Результат:

Как видно из примера, изображение было “стабилизировано”, то есть вернулось к позиции предыдущего кадра.

Однако, метод имеет два существенных недостатка:

1) При сильном отклонении изменённого изображения изменение позиций характерных точек вычисляется с погрешностью:

2) Метод работает тем хуже, чем больше характерных точек пришлось на подвижные объекты (люди, машины, листья деревьев и т.д.), поскольку оптический поток всего изображения будет отличаться от оптического потока локальных участков (участков с людьми, машинами и т.д.).

Сравнение характерных точек.

Альтернативным способом нахождения характерных точек одного изображения на другом является поиск и сравнение характерных точек обоих изображений.

Поиск и описание характерных точек:

```
//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);
```

Для сравнения, используется объект BFMatcher (сравнение прямым перебором):

```
//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
```

После того, как общие точки были найдены, необходимо отбросить все повторы (когда одна точка имеет более одного совпадения):

```
//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);
```

В результате, массив mask будет хранить данные о том, какие из найденных совпадений стоит использовать в дальнейших вычислениях.

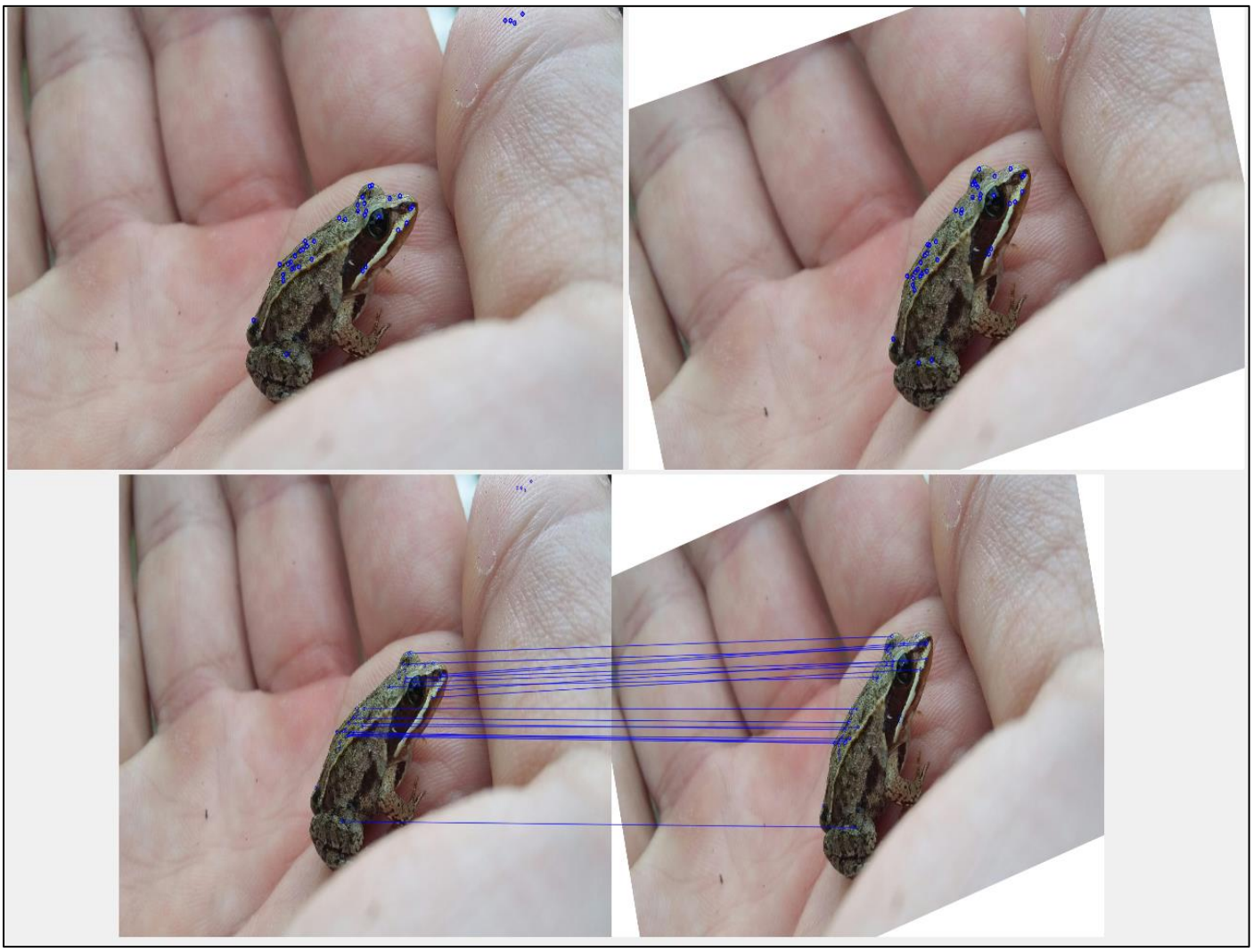
Дополнительно, можно провести отбор совпадений по параметрам масштабирования и поворота:

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

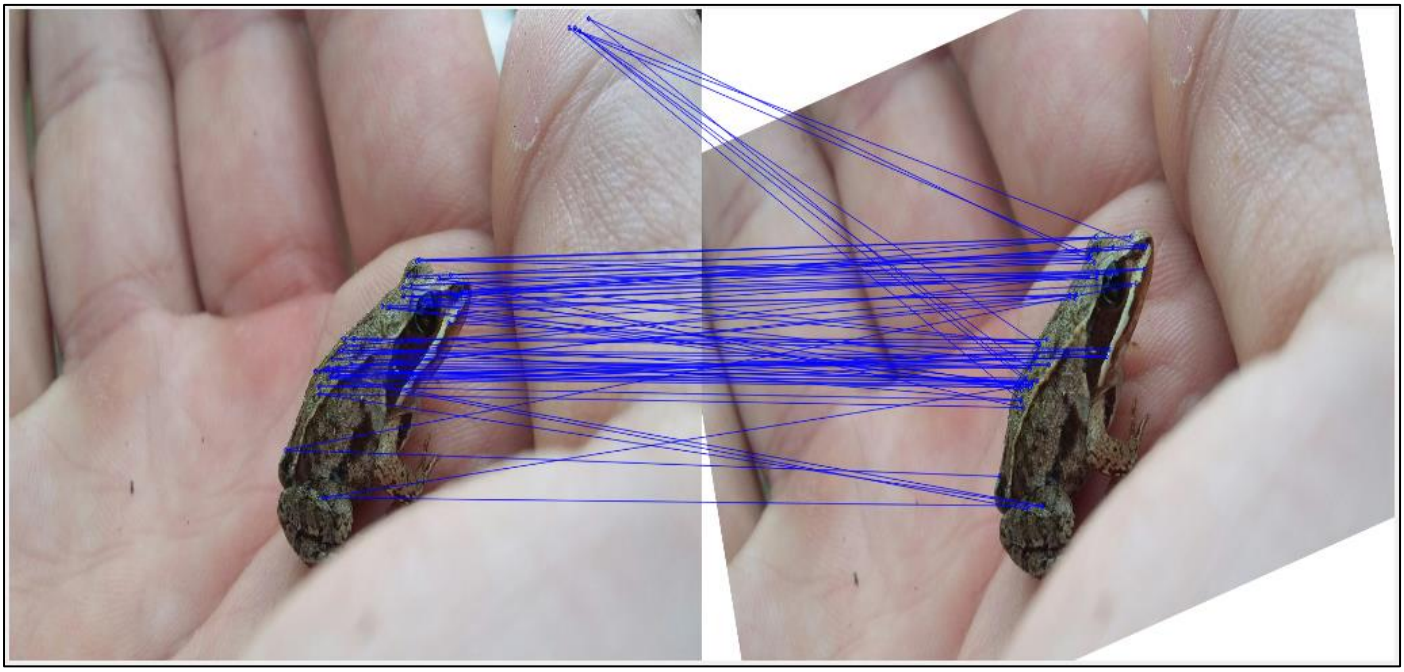
А затем, нарисовать оставшиеся совпадения при помощи встроенной функции:

```
Features2DToolbox.DrawMatches(twistedImg, GFP1, baseImg, GFP2, matches, res, new MCvScalar(255, 0,
0), new MCvScalar(255,0, 0), mask);
```

Результат:



Для примера, совпадения без тестов на уникальность, поворот и масштаб (все предполагаемые совпадения характерных точек):



Получить матрицу гомографической проекции по массиву совпадений можно следующим образом:

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

После чего, применить обратное преобразование и получить искомый результат:

Сравнение характерных точек может применяться и за пределами задачи стабилизации изображения. Например, имея изображения с примерами дорожных знаков, можно искать дорожные знаки на любом изображении.

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.