BIO.Framework tutoriál - 3D Face Recognition

Štěpán Mráček (imracek@fit.vutbr.cz)

28.10.2011

1 Úvod

Tento krátký tutoriál slouží jako pomůcka při vývoji vlastního biometrického systému s použitím knihovny BIO.Framework. Popisuje postup při implementaci jednoduchého rozpoznávání 3D obličejů podle výškové mapy obličeje. Databáze snímků je ke stažení na http://www.stud.fit.vutbr.cz/∼xmrace01/frgc.zip.

BIO.Framework je sada šablon a rozhraní v prostředí Microsoft .NET. Umožňuje implementovat vlastní stěžejní části biometrického systému a tyto jednotlivé části propojit v jeden fungující celek. V tomto dokumentu bude postupně popsán postup při tvorbě jednotlivých tříd (implementací rozhraní).

2 Rozpoznávání 3D obličejů podle histogramů

Metoda rozpoznávání obličejů použitá v tomto příkladu vychází z [1]. Princip extrace rysů je poměrně jednoduchý. Obličej je rozdělen na M vodorovných pruhů a v rámci každého pruhu jsou jednotlivé body rozděleny do K košů podle svých z-ových součadnic - v každém pruhu je tedy vytvořen histogram z-ových souřadnic jednotlivých bodů. Výsledný vektor rysů je pak tvořen jednotlivými hodnotami těchto histogramů. Ilustrace vstupního snímku a výsledného vektoru rysů je znázorněna na obrázku 1.

V našem příkladu používáme snímky 100x100 pixelů. Pro jednoduchost je rozdělíme na 10 vodorovných pruhů a v každém pruhu bude 10 košů ($K = 10$, $N = 10$). Podobnost dvou různých vektorů $\mathbf{x}$ a $\mathbf{y}$ porovnáme tak, že sečteme absolutní hodnoty rozdílů jednotlivých komponent vektorů rysů:

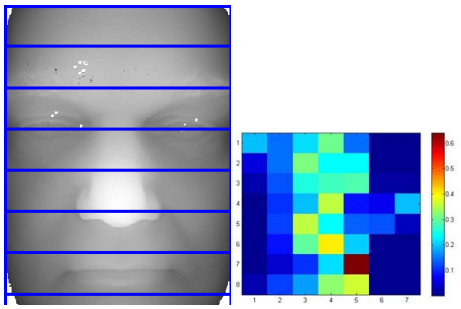
$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{K*N} |\mathbf{x}_i - \mathbf{y}_i|$$

3 Návrh systému

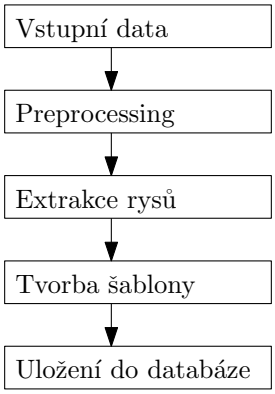
V reálném systému jsou často vstupní data ve formě jednotlivých bodů v prostoru, navíc obličej může být různě rotován. Pro potřeby tohoto příkladu už budou vstupní data částečně předzpracována – obličeje budou zarovnány do určité referenční pozice, převedeny do výškové mapy a uloženy ve formátu PNG.

Obecný postup, který se používá v biometrických systémech je na obrázku 2. Omezíme se především na extrakci rysů, protože vstupní snímek je již předzpracován a o tvorbu šablon a uložení do databáze se postará BIO.Framework.

Všechny vstupní snímky budou umístěny v adresáři D:\Data\FaceDB a jména souborů budou dodržovat následující konvenci:



Obrázek 1: Vstupní výšková mapa a odpovídající vektor rysů



Obrázek 2: Obecný potup při zpracování vstupního snímku, extrakci rysů a tvorbě referenční šablony

<id_osoby>-<pořadí_skenu>-small-range.png

Tedy například 2463-1-small-range.png. Úspěšnost rozpoznávání pak budeme testovat tak, že od každé osoby použijeme první sken jako referenční, vytvoříme z něj šablonu a uložíme do databáze. Všechny ostatní snímky pak použijeme pro testování.

4 Postup implementace

4.1 Extrakce rysů

Vstupní snímky jsou v odstínech šedi ve formátu PNG. K tomutu účelu použijeme již implementovanou třídu `EmguGray ImageInputData` ze jmenného prostoru `BIO.Framework. Extensions.Emgu.Input`. Výstupní vektor rysů je sada $N \cdot K = 10 \cdot 10 = 100$ čísel. K tomuto účelu použijeme třídu `EmguMatrixFeatureVector` ze jmenného prostoru `BIO.Framework. Extensions.Emgu. FeatureVector`. O extrakci rysů se stará třída `FaceFeatureVectorExtractor`, která implementuje rozhraní `IFeatureVectorExtractor`:

```
1  using System;
2  using System.Drawing;
3  using BIO.Framework.Core;
4  using BIO.Framework.Extensions.Emgu.FeatureVector;
5  using BIO.Framework.Extensions.Emgu.Input;
6
7  namespace BIO.Project.Example3DFaceRecognition
8  {
9      class FaceFeatureVectorExtractor : IFeatureVectorExtractor<EmguGrayImageInputData, EmguMatrixFeatureVector>
10     {
11         // implementovaná metoda rozrhaní IFeatureVectorExtractor
12         public EmguMatrixFeatureVector extractFeatureVector(EmguGrayImageInputData input)
13         {
14             // kontrola vstupu
15             if (input.Image.Width != 100 ||
16                 input.Image.Height != 100)
17                 throw new Exception("Image size has to be 100x100 pixels");
18
19             var processedImage = input.Image.SmoothGaussian(3);
20
21             // Výstupní vektor rysů
22             var featureVector = new EmguMatrixFeatureVector(new Size(1, 100));
23             featureVector.FeatureVector.SetZero();
24
25             var minInStripe = new double[10];
26             var maxInStripe = new double[10];
27             for (var i = 0; i < 10; i++)
28             {
29                 minInStripe[i] = double.MaxValue;
30                 maxInStripe[i] = -double.MaxValue;
31             }
32
33             // Výpočet minimální a maximální hodnoty v každém pruhu
34             for (var y = 0; y < 100; y++)
35             {
36                 for (var x = 0; x < 100; x++)
37                 {
38                     var stripe = y/10;
39                     var value = processedImage[y, x].Intensity;
40
41                     if (value < minInStripe[stripe])
42                         minInStripe[stripe] = value;
```

```
                    if (value > maxInStripe[stripe])
                        maxInStripe[stripe] = value;
                }
            }

            // Výpočet histogramu
            for (var y = 0; y < 100; y++)
            {
                for (var x = 0; x < 100; x++)
                {
                    var stripe = y / 10;
                    var value = processedImage[y, x].Intensity;

                    var normalizedValue =
                        ((double) (value - minInStripe[stripe]))/(maxInStripe[stripe] - minInStripe[stripe]);
                    var bin = (int) (normalizedValue*10);
                    if (bin >= 10) bin = 9;

                    var row = 10*stripe + bin;
                    featureVector.FeatureVector[row, 0] += 1;
                }
            }

            return featureVector;
        }
    }
}
```

4.2 Komparátor

O porovnávání vstupního vektůru rysů a šablony, která je uložena v databázi se stará komparátor. Implementuje rozhraní `IFeatureVectorComparator`. Toto rozhraní rozlišuje typ extrahovaného vektoru rysů a typ vektoru rysů ze šablony v databázi, protože obecně mohou být různá. V našem případě se ale jedná o týž datový typ. Komparátor poskytuje číselný údaj nakolik jsou vstupní vektory shodné/odlišné.

```
using System;
using BIO.Framework.Core.Comparator;
using BIO.Framework.Extensions.Emgu.FeatureVector;
namespace BIO.Project.Example3DFaceRecognition
{
    class FaceComparator : IFeatureVectorComparator<EmguMatrixFeatureVector, EmguMatrixFeatureVector>
    {
        public double computeMatchingScore(EmguMatrixFeatureVector extracted, EmguMatrixFeatureVector templated)
        {
            double sum = 0;

            if (extracted.FeatureVector.Size != templated.FeatureVector.Size ||
                extracted.FeatureVector.Cols != 1 || templated.FeatureVector.Cols != 1)
                throw new ArgumentException("Feature vector and template mismatch.");

            var n = extracted.FeatureVector.Rows;
            for (var i = 0; i < n; i++)
            {
                sum += Math.Abs(extracted.FeatureVector[i, 0] - templated.FeatureVector[i, 0]);
            }

            return sum;
        }
    }
}
```

4.3 Tvorba databáze

O tvorbu biometrické databáze ze souborů na disku se stará třída `FaceDatabaseCreator`. Třída využívá jíž implementovaných tříd `StandardRecord` a `StandardRecordData`.

```
using System; using System.IO;
using BIO.Framework.Core.Database;
using BIO.Framework.Extensions.Standard.Database.InputDatabase;

namespace BIO.Project.Example3DFaceRecognition
{
    class FaceDatabaseCreator : IDatabaseCreator<StandardRecord<StandardRecordData>>
    {
        readonly string _databasePath;

        public FaceDatabaseCreator(string databasePath)
        {
            _databasePath = databasePath;
        }

```

```
        public Database<StandardRecord<StandardRecordData>> createDatabase()
        {
            var database = new Database<StandardRecord<StandardRecordData>>();

            var di = new DirectoryInfo(_databasePath);
            var files = di.GetFiles("*-small-range.png");
            foreach (var f in files)
            {
                var parts = f.Name.Split(new[] { '-' }, StringSplitOptions.RemoveEmptyEntries);

                var bioID = new BiometricID(parts[0], "3d_face_range_image");
                var data = new StandardRecordData(f.FullName);
                var record = new StandardRecord<StandardRecordData>(bioID, data);

                database.addRecord(record);
            }

            return database;
        }
    }
}
```

Převod z konrétního záznamu v databázi na vstupní obrázek je definován následovně:

```
using BIO.Framework.Core;
using BIO.Framework.Extensions.Emgu.Input;
using BIO.Framework.Extensions.Standard.Database.InputDatabase;

namespace BIO.Project.Example3DFaceRecognition
{
    class FaceInputDataCreator : IInputDataCreator<StandardRecord<StandardRecordData>, EmguGrayImageInputData>
    {
        public EmguGrayImageInputData createInputData(StandardRecord<StandardRecordData> record)
        {
            return new EmguGrayImageInputData(record.BiometricData.Data);
        }
    }
}
```

4.4 Zakomponování jednotlivých částí do celku

Pro zakomponování jednotlivých částí do celku je třeba implementovat rozhraní `IBiometricAlgorithmSettings` ze jmenného prostoru `BIO.Framework.Core` a také implementovat abstraktní třídu `BiometricSystemFactory` ze jmenného prostoru `BIO.Framework.Core.BiometricSystem`.

Ve třídě `FaceRecognitionAlgorithmSetings` jsou zaregistrovány jednotlivé komponenty systému, které provádějí konrétní úlohy:

```
namespace BIO.Project.Example3DFaceRecognition
{
    class FaceRecognitionAlgorithmSetings :
        StandardSettings<
            StandardRecord<StandardRecordData>,
            EmguGrayImageInputData,
            EmguMatrixFeatureVector,
            Template<EmguMatrixFeatureVector>,
            EmguMatrixFeatureVector>
    {
        public FaceRecognitionAlgorithmSetings() : base("FaceRecognitionExample"){}

        protected override IInputDataCreator<StandardRecord<StandardRecordData>, EmguGrayImageInputData>
        createInputDataCreator()
        {
            return new FaceInputDataCreator();
        }

        protected override IFeatureVectorExtractor<EmguGrayImageInputData, EmguMatrixFeatureVector>
        createTemplateFeatureVectorExtractor()
        {
            return new FaceFeatureVectorExtractor();
        }

        protected override IFeatureVectorExtractor<EmguGrayImageInputData, EmguMatrixFeatureVector>
        createExtractedFeatureVectorExtractor()
        {
            return new FaceFeatureVectorExtractor();
        }

        protected override ITemplateCreator<EmguMatrixFeatureVector, Template<EmguMatrixFeatureVector>>
        createTemplateCreator()
        {
            return new TemplateCreator<EmguMatrixFeatureVector, Template<EmguMatrixFeatureVector>>();
        }

```

```
        protected override IComparator<EmguMatrixFeatureVector, EmguMatrixFeatureVector, Template<EmguMatrixFeatureVector>>
        createTemplateComparator()
        {
            return
                new Comparator<EmguMatrixFeatureVector, EmguMatrixFeatureVector, Template<EmguMatrixFeatureVector>>(
                    CreateFeatureVectorComparator(), CreateScoreSelector());
        }

        private static IFeatureVectorComparator<EmguMatrixFeatureVector, EmguMatrixFeatureVector>
        CreateFeatureVectorComparator()
        {
            return new FaceComparator();
        }

        private static IScoreSelector CreateScoreSelector()
        {
            return new MinScoreSelector();
        }
    }
}
```

Třída `FaceBiometricSystemFactory` následně zapouzdří vytvoření databáze, nastavení algoritmu a vyhodnocení:

```
namespace BIO.Project.Example3DFaceRecognition
{
    class FaceBiometricSystemFactory :
        BiometricSystemFactory<
            StandardRecord<StandardRecordData>,
            EmguGrayImageInputData,
            EmguMatrixFeatureVector,
            Template<EmguMatrixFeatureVector>,
            EmguMatrixFeatureVector,
            MemoryPersistentTemplate>
    {
        protected override IEvaluatorSettings<MemoryPersistentTemplate> createEvaluationSettings()
        {
            return new FaceEvaluationSettings();
        }
        protected override IBiometricAlgorithmSettings<
            StandardRecord<StandardRecordData>,
            EmguGrayImageInputData,
            EmguMatrixFeatureVector,
            Template<EmguMatrixFeatureVector>,
            EmguMatrixFeatureVector>
        createBiometricSettings()
        {
            return new FaceRecognitionAlgorithmSetings();
        }

        protected override IDatabaseCreator<StandardRecord<StandardRecordData>> createInputDatabaseCreator()
        {
            return new FaceDatabaseCreator(@"D:\Data\FaceDB");
        }
    }
}
```

4.5 Testování

```
using System;
using System.Collections.Generic;
using BIO.Framework.Core.Results.Visualization;
using BIO.Framework.Extensions.Standard.Database.InputDatabase;
using BIO.Framework.Extensions.Standard.Database.Subsets;
using BIO.Framework.Extensions.Standard.Results.Visualization;
using BIO.Framework.Extensions.ZedGraph.Results.Visualization;

namespace BIO.Project.Example3DFaceRecognition
{
    static class Program
    {
        [STAThread]
        static void Main()
        {
            var system = new FaceBiometricSystemFactory();
            var database = system.getInputDatabaseCreatorInstance().createDatabase();

            //create database subsets
            var templateTestSubset =
                new TemplateAndEvaluationDatabaseSubsetCreator<StandardRecord<StandardRecordData>>(database, 1);

            //template subset
            var templateDbSubset =
                templateTestSubset.getDatabaseSubset(
                    TemplateAndEvaluationDatabaseSubsetCreator<StandardRecord<StandardRecordData>>.TemplateSubset);
```

```
27              //test subset
28              var testDbSubset =
29                  templateTestSubset.getDatabaseSubset(
30                      TemplateAndEvaluationDatabaseSubsetCreator<StandardRecord<StandardRecordData>>.EvaluationSubset);
31
32              //debugging console output
33              Console.WriteLine("Database: Create reference templates");
34              var infol = new Framework.Utils.Console.Database.DatabaseOverview(templateDbSubset.getIterator());
35              infol.showGlobalOverview();
36
37              //database for templates
38              var templateDatabase = new BIO.Framework.Core.Database.BiometricDatabase.BiometricDatabase();
39              //progress report event
40              system.getEvaluator().getCompleteEvaluatorInstance().ProgressChangedEvent += ProgramProgressChanged;
41
42              //template extraction
43
44              Console.WriteLine("Template extraction");
45              system.getEvaluator().getCompleteEvaluatorInstance().extractTemplates(
46                  templateDbSubset.getIterator(),
47                  templateDatabase);
48
49              Console.WriteLine();
50              Console.WriteLine("Template extraction done");
51
52              //templates are ready - now evaluate db
53              Console.WriteLine("Algorithm evaluation");
54              //where to store results
55              var results = new Framework.Core.Results.Results();
56              //own evaluation
57              system.getEvaluator().getCompleteEvaluatorInstance().evaluateRecords(
58                  testDbSubset.getIterator(),
59                  templateDatabase,
60                  results);
61
62              Console.WriteLine();
63              Console.WriteLine("Algorithm evaluation done");
64
65              //postprocess results
66              var postprocessors = new List<IResultsVisualizer>
67                                       {
68                                           new StatisticsSummaryResultsPostprocessor("summary.txt"),
69                                           new ZedGraphResultsGraphVisualizer("GenuineImpostor.png")
70                                       };
71
72              //statistics
73              foreach (var pp in postprocessors)
74              {
75                  pp.ProgressChangedEvent += ProgramProgressChanged;
76                  pp.postprocessResults(results);
77              }
78              Console.WriteLine();
79
80              //show results in Windoows GUI form
81              var resultsForm = new Framework.Utils.UI.Results.ResultsForm(results);
82              System.Windows.Forms.Application.Run(resultsForm);
83          }
84
85          static void ProgramProgressChanged(Framework.Core.ProgressReport progress)
86          {
87              Console.Write("\r{1}% {0}", progress.Message, progress.Progress);
88          }
89      }
90  }
```

Reference

[1] X. Zhou, H. Seibert, C. Busch, and W. Funk. A 3d face recognition algorithm using histogram-based features. *Eurographics Workshop on 3D Object Retrieval*, pages 65–71, 2008.