Face Recognition using 2D Fast Fourier Transform

This article explains how to implement a simple face recognition system based on image analysis using the Fourier spectrum. Recognition is done by finding the closest match between feature vectors containing the Fourier coefficients at selected frequencies. The introduced method well compares to other competing approaches.

Note that even though the discussion is fairly complex, there is no need to be frightened off by this technique; the article delivers (and uses) tools which make the process easy for other developers to work with. Will be shown how to implement it in Windows Phone 8 creating a camera app through the new feature Lenses.

Note: This article was a winner in the Windows Phone 8 Wiki Competition 2012Q4.

Introduction

No two human faces are identical, which makes them well suited for use in identification and access control applications - the obvious advantage over competing identity methods is that face recognition doesn't require physical interaction for access - it only needs the subject to look into a camera.

Automated face recognition systems have generally evolved along two main routes, either the analysis of grey level information (often called "template based") or the extraction of mainly geometrical features such as shape, profile or hair colour. Humans are thought to view faces primary in a holistic manner and experiments suggest that holistic approaches are superior to geometrical recognition systems.

The work presented here comprises a novel template based approach that compares very well to other more complex methods that are used commonly such Hidden Markov Models or back propagation Neural Network. The technique is based on the Fourier spectrum of facial images, thus it relies on a global transformation: every pixel in the image contributes to each value in the spectrum.

The Fourier spectrum is a plot of the energy against spatial frequencies, where spatial frequencies relate to the spatial relations of intensities in the image. In our case this translates to distances between areas of particular brightness such as the overall area of the head or the distance between the eyes. Higher frequencies describe finer details and contrary to what you might think we found them less useful for identification, just as humans can recognize a face from a brief look without focusing on small details.

The recognition of faces is done by finding the closest match (the difference or distance) between the newly presented face and all those faces known to the system. The distances are calculated between the feature vectors with entries that are the Fourier transform values at specially chosen frequencies. As few as 30 frequencies yield excellent results.

Fast Fourier Transform

The Fourier Transform is an important image processing tool which is used to decompose an image into its sine and cosine components. The output of the transformation represents the image in the Fourier or frequency domain, while the input image is the spatial domain equivalent. In the Fourier domain image, each point represents a particular frequency contained in the spatial domain image.

The Fourier Transform is used in a wide range of applications, such as image analysis, image filtering, image reconstruction and image compression, text orientation finding that will be covered (hopefully) in further articles.

The Fast Fourier Transform (FFT) is an efficient algorithm to compute the discrete Fourier transform (DFT) and its inverse. The discrete Fourier transform (DFT) transforms one function into another, which is called the frequency domain representation of the original function. The DFT requires an input function that is discrete. Such inputs are often created by sampling a continuous function, such as a person's voice. The discrete input function must also have a limited (finite) duration, such as one period of a periodic sequence or a windowed segment of a longer sequence.

The DFT is the sampled Fourier Transform and therefore does not contain all frequencies forming an image, but only a set of samples which is large enough to fully describe the spatial domain image. The number of frequencies corresponds to the number of pixels in the spatial domain image, i.e. the image in the spatial and Fourier domain are of the same size.

For a square image of size N×N, the two-dimensional DFT is given by:

$$F(k,l) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j)\, e^{-\imath 2\pi(\frac{ki}{N}+\frac{lj}{N})}$$

where f(a,b) is the image in the spatial domain and the exponential term is the basis function corresponding to each point F(k,l) in the Fourier space. The equation can be interpreted as: the value of each point F(k,l) is obtained by multiplying the spatial image with the corresponding base function and summing the result.

The Fourier Transform produces a complex number valued output image which can be displayed with two images, either with the real and imaginary part or with magnitude and phase. In image processing, often only the magnitude of the Fourier Transform is displayed, as it contains most of the information of the geometric structure of the spatial domain image. However, if we want to re-transform the Fourier image into the correct spatial domain after some processing in the frequency domain, we must make sure to preserve both magnitude and phase of the Fourier image.

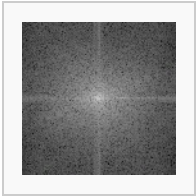
The Fourier domain image has a much greater range than the image in the spatial domain. Hence, to be sufficiently accurate, its values are usually calculated and stored in float values.

(C) Copyright Nokia 2014. All rights reserved.

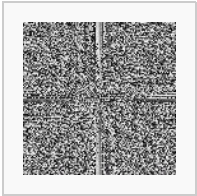
The Fourier Transform is used if we want to access the geometric characteristics of a spatial domain image. Because the image in the Fourier domain is decomposed into its sinusoidal components, it is easy to examine or process certain frequencies of the image, thus influencing the geometric structure in the spatial domain.

In most implementations the Fourier image is shifted in such a way that the DC-value (i.e. the image mean) F(0,0) is displayed in the centre of the image. The further away from the centre an image point is, the higher is its corresponding frequency.

Original image



Magnitude



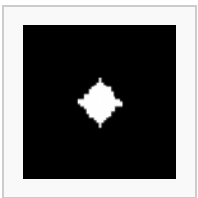
Phase

For a slightly deeper view, see Sound pattern matching using Fast Fourier Transform in Windows Phone.

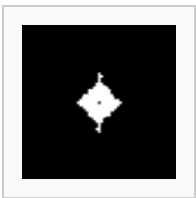
Selecting Frequencies

From the spectrum it can be seen that almost all the information is contained near the center, within the low frequencies. Thus it seems reasonable that these frequencies will also provide the best ground for the recognition process. Valuable frequencies don't lie in a circle around the origin but more in a rhombus shaped region.

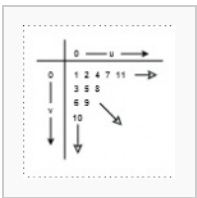
We know that the second half of FFT carry no useful and duplicated information, so we can half the data to treat. As the 2D FFT is built as two pass of 1D FFT it means that we can focus just to one quadrant reducing further the data to treat.



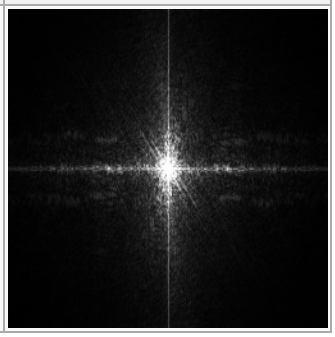
Most variant frequencies (Real part)

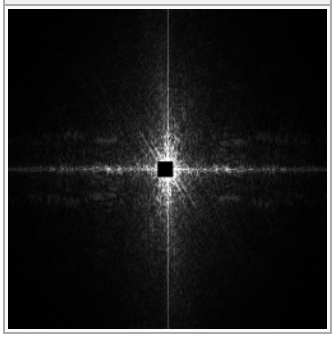
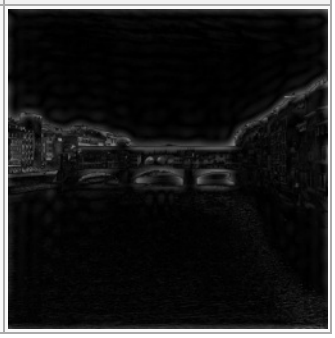


Most variant frequencies (Imaginary part)



Selected numbering

Original Image	Fourier Transform
	

Deleting low frequencies (the most important)	Result
	

As you can see, very few elements brings great part of information. This is the reason why we use the center part of FFT.

Deleting high frequencies	Result

(C) Copyright Nokia 2014. All rights reserved.

Deleting a great part of information related to high frequencies the effect is simply a little blurring, but great part of image is maintained.
For your information this is the basis of image compression and the reason why increasing compression you get the blurring effect on your photos.

Working with 2D FFT in Windows Phone 8

A new great Windows Phone's feature is Camera Lenses that enables you to build apps into the camera app. Users can launch “Lenses” apps directly from the camera app making the facial recognition process a perfect solution for Lenses application.

I provided some tips I learned and found useful developing a lens application to save time during development.

- Download Dsp.zip and extract DSP.cs from the zip. Then add it into your project. **DSP.cs** provides a namespace called `DSP` and a class `FourierTransform` containing a set of functions to compute the FFT.

Don't forget to include the namespace DSP

```
using DSP;
```

Inside the **WMAppManifest.xml** file add the following capabilities:

```
<Capabilities>
          <Capability Name="ID_CAP_ISV_CAMERA" />
          <Capability Name="ID_CAP_MEDIALIB_PHOTO" />
</Capabilities>

<Extensions>
          <Extension ExtensionName="Camera_Capture_App" ConsumerID="{5B04B775-356B-4AA0-
AAF8-6491FFEA5631}" TaskID="_default" />
</Extensions>
```

1. Each `Extention` element describes an App Connect extension and the `Extention` tag must be allocated after `Tokens` tag.
2. `ExtensionName` ss the identifier for the type of extension support. The value is `Camera_Capture_App`
3. `ConsumerID` restricts access to the extension to the consumer with the specified ProductID. All search extensions require the same value, **5B04B775-356B-4AA0-AAF8-6491FFEA5661**.

Now your application is registered as Lenses app and can be found and called from the main Camera app.

To display the Camera flow into your application let's add the following code:

```
<Button Content="Snap Picture" Click="SaveImage" />
<Grid x:Name="ContentPanel" Grid.Row="1" >
         <Grid.Background>
                  <VideoBrush x:Name="viewfinderBrush" />
         </Grid.Background>
</Grid>
```

Now let's begin to build the app.

```
using Microsoft.Devices; // Needed for PhotoCamera
using Microsoft.Xna.Framework.Media;
using System.Windows.Media.Imaging;
using Microsoft.Phone; // Needed for PictureDecoder

namespace Face_Recognition
{
```


(C) Copyright Nokia 2014. All rights reserved.

```
    public partial class MainPage : PhoneApplicationPage
    {
        PhotoCamera cam;
        MediaLibrary library = new MediaLibrary();

        public static WriteableBitmap CapturedImage;
        private static int W = 256;
        private static int H = 256;
        private static int matchSamples = 25;

        private double[] compareSignal = new Double[matchSamples];

        private Double[] pRealIn = new Double[W * H];
        private Double[] pImagIn = new Double[W * H];
        private Double[] pRealOut = new Double[W * H];
        private Double[] pImagOut = new Double[W * H];
        private Double[] pRealOut2 = new Double[W * H];
        private Double[] pImagOut2 = new Double[W * H];

        public MainPage()
        {
                InitializeComponent();

                this.Loaded += Lense_Loaded;
        }

        void Lense_Loaded(object sender, RoutedEventArgs e)
        {
              if (PhotoCamera.IsCameraTypeSupported(CameraType.FrontFacing))
              {
                cam = new Microsoft.Devices.PhotoCamera(CameraType.FrontFacing);
                cam.CaptureImageAvailable += cam_CaptureImageAvailable;
                viewfinderBrush.SetSource(cam);
              } else
              if (PhotoCamera.IsCameraTypeSupported(CameraType.Primary))
              {
                cam = new Microsoft.Devices.PhotoCamera(CameraType.Primary);

                cam.CaptureImageAvailable += cam_CaptureImageAvailable;
                viewfinderBrush.SetSource(cam);
              }
        }

}
```

Converting a pixel to Grayscale

Here a useful function to convert a colored pixel into gray-scale. That operation allow us to save more computation,

```
        internal int ColorToGray(int color)
        {
            int gray = 0;

            int a = color >> 24;
            int r = (color & 0x00ff0000) >> 16;
            int g = (color & 0x0000ff00) >> 8;
            int b = (color & 0x000000ff);

            if ((r == g) && (g == b))
            {
                gray = color;
            }
            else
            {
                // Calculate for the illumination.
                // I =(int)(0.109375*R + 0.59375*G + 0.296875*B + 0.5)
                int i = (7 * r + 38 * g + 19 * b + 32) >> 6;
```


(C) Copyright Nokia 2014. All rights reserved.

```
                gray = ((0x1) << 24) | ((i & 0xFF) << 16) | ((i & 0xFF) << 8) | (i &
0xFF);
            }
            return gray;
        }
```

FFT 2D

Tip: Don't create a `BitmapImage` or `WriteableBitmap` on the UI thread because they are `DependencyObjects`. If you do you will get the `ObjectDisposedException` error because the stream for the image is closed already when the dispatcher handles your request. Move into your Dispatcher invocation.

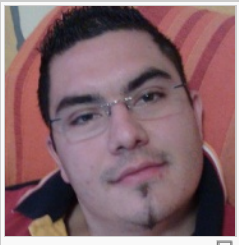
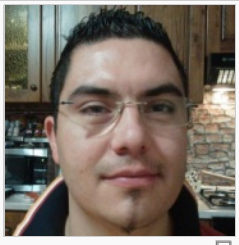
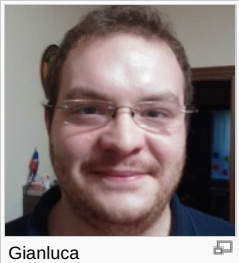
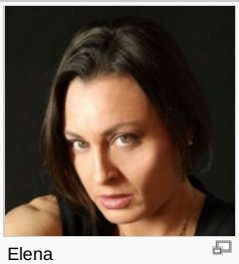
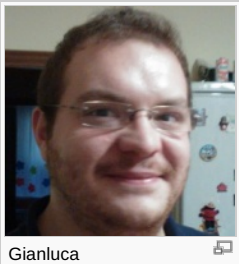
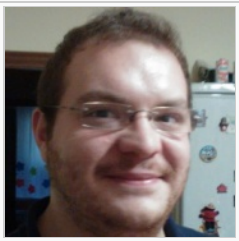
```
void cam_CaptureImageAvailable(object sender, ContentReadyEventArgs e)
{
            Deployment.Current.Dispatcher.BeginInvoke(delegate()
            {
                //Take JPEG stream and decode into a WriteableBitmap object
                MainPage.CapturedImage = PictureDecoder.DecodeJpeg(e.ImageStream,W,H);

                //Collapse visibility on the progress bar once writeable bitmap is
visible.
                progressBar.Visibility = Visibility.Collapsed;

                int[] pixel = MainPage.CapturedImage.Pixels;

                int color = 0;
                for (int y = 0; y < MainPage.CapturedImage.PixelHeight; y++)
                {
                                for (int x = 0; x < MainPage.CapturedImage.PixelWidth;
x++)
                                {
                                                color = MainPage.CapturedImage.Pixels[x
+ (y * MainPage.CapturedImage.PixelWidth)];
                                                pRealIn[x + (y *
MainPage.CapturedImage.PixelWidth)] = DSP.Utilities.ColorToGray(color) & 0xFF;
                                }
                }

                Double[] match;
                double mse = 0;

                System.Diagnostics.Debug.WriteLine("Using Fourier");
                DSP.FourierTransform.Compute2D((uint)W, (uint)H, ref pRealIn, null, ref
pRealOut, ref pImagOut, false);
                match = DSP.Utilities.triangularExtraction(ref pRealOut, (uint)W,
(uint)H, (uint)matchSamples, 0);
                mse = DSP.Utilities.MSE(ref compareSignal, ref match,
(int)matchSamples);

                // normalize
                mse /= 1000000000;

                mseResult.Text = "" + Math.Round(mse);
                System.Diagnostics.Debug.WriteLine("MSE:" + mseResult.Text);

                // Just as Demo we save the current image into a buffer called match in
order to compare it with the next image. In real life situations the sample to compare
is aved into a file or database.
                Array.Copy(match, 0, compareSignal, 0, matchSamples);

                // Sample code to reconstruct the image from FFT spectrum
                color = 0;
                for (int y = 0; y < MainPage.CapturedImage.PixelHeight; y++)
                {
                                for (int x = 0; x < MainPage.CapturedImage.PixelWidth;
x++)
                                {
```


(C) Copyright Nokia 2014. All rights reserved.

```
                                                    color = (int)Math.Floor(pRealIn[x + (y
* MainPage.CapturedImage.PixelWidth)]);
                                                   color = (color > 0) ? ((color > 255) ?
255 : color) : 0;
                                                   MainPage.CapturedImage.Pixels[x + (y *
MainPage.CapturedImage.PixelWidth)] = (1 << 24) | (color << 16) | (color << 8) | color;
                                 }
                }
            });
 }
```

Results

The table below shows a comparison between a number of images. If the result value is lower the images are a better match - in this case we would set the `threshold` to *50* and any result less would indicate a correct match

Image A	Image B	Result
Giuseppe	 Giuseppe	37
 Giuseppe	 Gianluca	61
Elena	Elena	36
 Elena	 Gianluca	618
 Gianluca	Elk	386

(C) Copyright Nokia 2014. All rights reserved.

Elk	Elk	34

It is interesting to note how the algorithm is fairly tolerant of rotated faces. Also notice the how much smaller the difference is between my two brothers *Giuseppe* and *Gianluca* compared to *Elena* with *Gianluca*. I find it amusing that my brother *Gianluca* looks more like an *Elk* than *Elena* :-)

It is important to understand that results are strongly influenced by the background, how the face is rotated or light conditions when compared to the sample image. The best case is when we manage photos with flat background as for the *Elena*'s case.

Use case

- Enabling wallet payment by face recognition

Summary

This article as shown a possible approach to solving the face recognition problem. Of course is not the ultimate solution as other more complex methods are available for example for military or national security use. It is however a very powerful, solution that is relatively easy to implement on mobile devices.

A lot of improvements will come for better results :

- Skin detection
- Skin quantization
- Hair detection
- Hair quantization

(C) Copyright Nokia 2014. All rights reserved.