University of Ljubljana

Faculty of Electrotechnical Engineering

Vitomir Štruc

The INface toolbox v2.0
The Matlab Toolbox for Illumination Invariant Face Recognition

Toolbox description and user manual

Ljubljana, 2011

Thanks

I would like to thank Dr. Peter Kovesi and Dr. Gabriel Peyré for the permission to use some of their functions in the toolbox.

I would also like to thank Alpineon Ltd and the Slovenian Research Agency (ARRS) for funding the postdoctoral project BAMBI (Biometric face recognition in ambient intelligence environments), which made the INFace toolbox v2.0 possible.

Foreword

The INFace (**I**llumination **N**ormalization techniques for robust **Face** recognition) toolbox v 2.0 is a collection of Matlab functions and scripts intended to help researchers working in the field of face recognition. The toolbox was produced as a byproduct of my research work and is freely available for download.

The INface toolbox v2.0 includes implementations of the following photometric normalization techniques: the single-scale-retinex algorithm, the multi-scale-retinex algorithm, the single-scale self quotient image, the multi-scale self quotient image, the homomorphic-filtering-based normalization technique, a wavelet-based normalization technique, a wevelet-denoising-based normalization technique, the isotropic-diffusion-based normalization technique, the anisotropic-diffusion-based normalization technique, the non-local-means-based normalization technique, the adaptive non-local-means-based normalization technique, the DCT-based normalization technique, a normalization technique based on steerable filters, a modified version of the anisotropic-diffusion-based normalization technique, the Gradientfaces approach, the Weberfaces approach, the multi-scale Weberfaces approach, the Tan and Triggs normalization technique and the large and small scale features normalization technique.

In addition to the listed techniques there are also a number of histogram manipulation functions included in the toolbox, which can be useful for the task of illumination invariant face recognition.

This document describes the basics of the toolbox, from installation to usage. The reader is referred to the original papers for more information on the theory underlying the individual techniques.

Contents

1. Installing the toolbox

The INface toolbox comes compressed in a ZIP archive or TAR ball. Before you can use it you first have to uncompress the archive into a folder of your choice. Once you have done that a new folder named *INface_tool* should appear and in this folder seven additional directories should be present, namely, *auxilary*, *histograms*, *photometric*, *mex*, *demos*, *postprocessors* and *other*. In most of these folders you should find a *Contents.m* file with a list and short descriptions of Matlab functions that should be featured in each of the seven folders.

1.1 Installation using the supplied script

When you are ready to install the toolbox, run Matlab and change your current working directory to the directory *INface_tools*, or in case you have renamed the directory, to the directory containing the files of the toolbox. Here, you just type into Matlabs command prompt:

```
install_INface
```

The command will trigger the execution of the the corresponding install script, which basically just adds all directories of the toolbox to Matlabs search path and compiles the C/C++ code contained in the *mex* directory. The installation script will produce so-called MEX (Matlab executable) files, which can be called from Matlab. The installation script was tested with Matlab version 7.5.0.342 (R2007b) on a 32-bit WindowsXP (SP3) OS installation as well as Matlab version 7.11.0.584 (R2010b) on a 64-bit Windows 7 installation. Version 2.0 of the toolbox ships with precompiled 64-bit MEX files.

Note that the install script was not tested on Linux machines. Nevertheless, I see no reason why the toolbox should not work on Linux as well. The only difficulty could be the installation of the toolbox due to potential difficulties with the path definitions. In case the install script fails (this applies for Linux and Windows users alike), you can perform the necessary steps manually as described in the next section.

1.2 Manual installation

The installation of the toolbox using the provided script can sometimes fail. There are two possible reasons I can foresee (this does not imply that there can't be others as well):

- you have not run the mex setup yet, or
- there is some difficulty with Matlabs search path.

If the installation fails due to the first error, this means that you have not selected an appropriate compiler for compiling the C/C++ code. To resolve this issue just type

```
mex -setup
```

into Matlabs command prompt and select a compiler from the provided list. Once this is done you can try to run the install script again. The script should now successfully compile the C/C++ code.

If the script fails due to path related issues, try adding the path to the toolbox folder and corresponding subfolders manually. In Matlabs main command window choose:

`File` → `Set Path` → `Add with Subfolders`.

When a new dialogue window appears navigate to the directory containing the toolbox, select it and click `OK`. Choose `Save` in the *Set Path* window and then click `Close`. This procedure adds the necessary paths to Matlabs search path. If you have followed all of the above steps, you should have successfully installed the toolbox and are ready to use it.

If you are attempting to add the toolbox folders and subfolders to Matlabs search path manually, make sure that you have administrator (or root) privileges, since these are commonly required for changing the `pathdef.m` file, where Matlabs search paths are stored.

1.3 Validating the installation

The INface toolbox v2.0 features a new script not present in the previous versions of the toolbox, which is aimed at validating the installation process of the

toolbox. This may come in handy if you encountered errors or warnings during the automatic installation process or attempted to install the toolbox manually. The script should be run from the base directory of the toolbox to avoid possible name clashes.

To execute the validation script type the following command into Matlabs command window:

```
check_install
```

The script will test all (or most) techniques contained in the toolbox. This procedure may take quite some time depending on the speed and processing power of your machine, so please be patient. If the installation was successful, you should see a report similar to the one below:

```
|============================================================|

VALIDATION REPORT:

Function ‘‘adjust_range’’ is working properly.
Function ‘‘gamma_correction’’ is working properly.
Function ‘‘threshold_filtering’’ is working properly.
Function ‘‘normalize8’’ is working properly.
Function ‘‘fitt_distribution’’ is working properly.
Function ‘‘rank_normalization’’ is working properly.
Function ‘‘histtruncate’’ is working properly.
Function ‘‘robust_postprocessor’’ is working properly.
Function ‘‘dog’’ is working properly.
Function ‘‘single_scale_retinex’’ is working properly.
Function ‘‘tantriggs’’ is working properly.
Function ‘‘weberfaces’’ is working properly.
Function ‘‘multi_scale_weberfaces’’ is working properly.
Function ‘‘gradientfaces’’ is working properly.
Function ‘‘anisotropic_smoothing_stable’’ is working properly.
Function ‘‘DCT_normalization’’ is working properly.
Function ‘‘lssf_norm’’ is working properly.
Function ‘‘adaptive_nl_means_normalization’’ is working properly.
Function ‘‘adaptive_single_scale_retinex’’ is working properly.
Function ‘‘anisotropic_smoothing’’ is working properly.
Function ‘‘isotropic_smoothing’’ is working properly.
Function ‘‘multi_scale_retinex’’ is working properly.
```

```
Function ‘‘single_scale_self_quotient_image’’ is working properly.
Function ‘‘multi_scale_self_quotient_image’’ is working properly.
Function "nl_means_normalization’’ is working properly.
Function ‘‘steerable_gaussians’’ is working properly.
Function ‘‘wavelet_denoising’’ is working properly.
Function ‘‘wavelet_normalization’’ is working properly.
Function ‘‘homomorphic’’ is working properly.

|============================================================|

SUMMARY:

All functions from the toolbox are working ok.

|============================================================|
```

In case the installation has (partially) failed, the report will explicitly show, which functions are not working properly. In most cases the toolbox should work just fine. Problems may occur only as a consequence of the C/C++ code compiling process.

Note that the `check_install` script was not initially meant to be an install validation script. The script was written to test whether all functions from the toolbox work correctly (i.e., with different combinations of input arguments). Basically, it was used for debugging the toolbox. However, due to the fact that the script tests all (or most) functions from the toolbox, it can equally well serve as an installation validation script.

2. Acknowledging the Toolbox

Any paper or work published as a result of research conducted by using the code in the toolbox or any part of it must include the following two publications in the reference section:

V. Štruc and N Pavešić, "Photometric normalization techniques for illumination invariance", In: Y.J. Zhang (Ed.), Advances in Face Image Analysis: Techniques and Technologies. IGI Global, 2011, pp. 279–300.

and

V. Štruc and N. Pavešić, "Gabor-Based Kernel Partial-Least-Squares Discrimination Features for Face Recognition", Informatica (Vilnius), vol. 20, no. 1, pp. 115–138, 2009.

BibTex files for the above publications are contained in the *other* folder and are stored as 'ACKNOWL1.bib' and 'ACKNOWL2.bib', respectively.

3. Toolbox description

The functions and scripts contained in the toolbox were produced as a byproduct of my research work. I have added a header to each of the files containing some examples of the usage of the functions and a basic description of the functions functionality. However, I made no effort in optimizing the code in terms of speed and efficiency. I am aware that some of the implementations could be significantly speeded up, but unfortunately I have not yet found the time to do so. I am sharing the code contained in the toolbox to make life easier for researcher working in the field of face recognition and students starting to get familiar with face recognition and its challenges. My main motivation for producing this toolbox was the fact that after quite some googling I have had no luck in finding any source code for illumination invariant face recognition, or better said, I have had no luck in finding source code where (in opinion) the implementation of the illumination normalization technique was correct.

The INFace (**I**llumination **N**ormalization techniques for robust **Face** recognition) toolbox in its current form is a collection of functions which perform illumination normalization and, hence, tackle one of the greatest challenges in face recognition. It should be noted that most of the techniques in this toolbox are implementations of so-called photometric normalization techniques. With the term photometric normalization technique we denote any normalization technique which performs illumination normalization at the preprocessing level (as opposed to techniques compensating for illumination induced appearance changes at the modeling or classification level). In case the reader is looking for implementations of model based approaches (such as the illumination cone models or spherical harmonics) this toolbox is not the right place to look for.

All techniques in the toolbox are implemented for use with grey-scale images and were tested on images of size 128×128 pixels. The default parameters of the techniques were chosen in such a way that *good* normalization results were obtained on images of this size, i.e., 128×128 pixels. Of course the term *good* is relative. The techniques can also be used with color images; however, in this case the reader is encouraged to write his own script to process the color images, e.g., component-wise.

The toolbox contains seven folders, named, *auxilary*, *histograms*, *photometric*, *postprocessors*, *mex*, *demos* and *other*. In the remainder of this chapter we will focus on the description of the contents of each of these folders.

3.1 The *photometric* folder

The folder named *photometric* is the main folder of the toolbox and contains the implementations of 21 different photometric normalization techniques proposed in the literature, i.e.:

- the single scale retinex algorithm (v2.0),
- the multi scale retinex algorithm (v2.0),
- the adaptive single scale retinex algorithm (v2.0),
- the homomorphic filtering based normalization technique (v2.0),
- the single scale self quotient image (v2.0),
- the multi scale self quotient image (v2.0),
- the DCT based normalization technique (v2.0),
- the wavelet based normalization technique (v2.0),
- the wavelet-denoising based normalization technique (v2.0),
- the isotropic diffusion based normalization technique (v2.0),
- the anisotropic diffusion based normalization technique (v2.0),
- the steerable filter based normalization technique (v2.0),
- the non-local means based normalization technique (v2.0),
- the adaptive non-local means based normalization technique (v2.0),
- the modified anisotropic diffusion based normalization technique (v2.0),
- the Gradientfaces based normalization technique (v2.0),
- the single scale Weberfaces normalization technique (v2.0),
- the multi scale Weberfaces normalization technique (v2.0),

- the large and small scale features based normalization technique (v2.0),
- the Tan and Triggs normalization technique (v2.0), and
- the DoG filtering based normalization technique (v2.0).

A basic description of the functionality of the function implementing the techniques in the toolbox together with a few examples is given below.

3.1.1 The single scale retinex algorithm

The single scale retinex (SSR) algorithm was proposed by Jobson et al. in [7]. Like the majority of photometric normalization techniques it is based on the so-called *retinex* theory which is explained in [8] in more detail. The SSR technique is implemented in the toolbox with a function that has the following prototype:

`[R,L] = single_scale_retinex(X,hsiz, normalize)`.

Here, `X` denotes the grey-scale image and `hsiz` stands for the parameter that controls the bandwidth of the Gaussian filter needed for the SSR technique. In the original paper by Jobson et al., the parameter `hsiz` is denoted as `c`. The input argument `normalize`$\in \{1, 0\}$ stands for a parameter controlling whether a postprocessing procedure is applied to the normalized image or not. The function returns the "illumination invariant" reflectance `R` and the estimated luminance function `L` (both in the log domain). Note here that the luminance function is returned only for visualization purposes, as it is usually only of little value from the perspective of illumination invariant face recognition.

If you type

```
help single_scale_retinex
```

you will get additional information on the function together with several examples of usage.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
[R,L]=single_scale_retinex(X);
figure,imshow(X,[]);
figure,imshow(R,[]);
```

```
figure,imshow(L,[]);
```

The code reads the image named `sample_image.bmp` into the variable `X` and applies the SSR algorithm with default parameter values to the image. After the procedure, all three images, i.e., the original image, the illumination invariant reflectance and the estimated luminance, are displayed in three separate figures.

We have applied the SSR technique to several images from the YaleB database. The results of the processing are shown in Fig. 3.1.

Figure 3.1: Sample images processed with the example code: original images (upper row), SSR processed images - the reflectance functions (lower row)

3.1.2 The multi scale retinex algorithm

The multi scale retinex (MSR) algorithm is an extension of the single scale retinex algorithm again proposed by Jobson et al. [6]. The MSR technique is implemented in the toolbox with a function that has the following prototype:

`Y = multi_scale_retinex(X,hsiz,normalize)`.

Here, `X` denotes the grey-scale image and `hsiz` stands for a vector of parameters that control the bandwidth of the Gaussian filters needed for the MSR technique. In the original paper by Jobson et al., the parameters in `hsiz` are denoted as `c`. The input argument `normalize`$\in \{1, 0\}$ stands for a parameter controlling whether a postprocessing procedure is applied to the normalized image or not.

If you type

```
help multi_scale_retinex
```

you will get additional information on the function together with several examples of usage.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
Y=multi_scale_retinex(X,[7 15 21]);
figure,imshow(X);
figure,imshow(Y,[]);
```

The code reads the image named `sample_image.bmp` into the variable `X` and applies the MSR algorithm with three filter scales (defined by the values in `hsiz`) to the image. After the procedure, both images, i.e., the original one and the processed one, are displayed in two separate figures.

We have applied the code to several images from the YaleB database. The results of the processing are shown in Fig. 3.2.

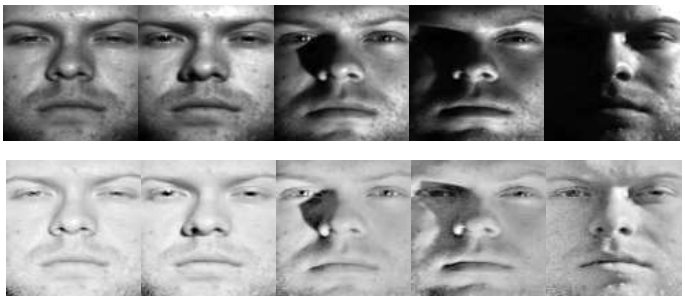

Figure 3.2: Sample images processed with the example code: original images (upper row), MSR processed images (lower row)

3.1.3 The adaptive single scale retinex algorithm

The adaptive single scale retinex (ASR) algorithm is one the newest additions to the retinex techniques and was proposed by Park et al. in [9]. The ASR technique is implemented in the toolbox with a function that has the following prototype:

`[R,L] = adaptive_single_scale_retinex(X,T,S,h,normalize)`.

Here, `X` denotes the input grey-scale image to be processed, `T` stands for the number of iterations performed during the processing, and `S` and `h` represent parameters needed by the technique. The input argument `normalize`$\in \{1, 0\}$

stands for a parameter controlling whether a postprocessing procedure is applied to the normalized image or not. The reader is referred to the original paper for more information on the parameters [9]. The function returns the "illumination invariant" reflectance `R` and the estimated luminance function `L` (both in the log domain). Note here that the luminance function is returned only for visualization purposes, as it is usually only of little value from the perspective of illumination invariant face recognition.

If you type

```
help adaptive_single_scale_retinex
```

you will get additional information on the function together with several examples of usage.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
[R,L] = adaptive_single_scale_retinex(X,15);
figure,imshow(X);
figure,imshow(R,[]);
figure,imshow(L,[]);
```

The code reads the image named `sample_image.bmp` into the variable `X` and applies the ASR algorithm with 15 iterations to the image. The parameters `S` and `h` are determined automatically as suggested by the authors of the technique, but could also be set arbitrary. After the execution of the code, all three images, i.e., the original image, the illumination invariant version (i.e., the reflectance) and the estimated luminance function, are displayed in three separate figures.

We have applied the above code to several images from the YaleB database. The results of the processing are shown in Fig. 3.3.

3.1.4 The homomorphic filtering based normalization technique

Homomorphic filtering (HOMO) is a well known normalization technique where the input image is first transformed into the logarithm and then into the frequency domain. Here, the high frequency components are emphasized and the low-frequency components are reduced. As a final step the image is transformed back into the spatial domain by applying the inverse Fourier transform and

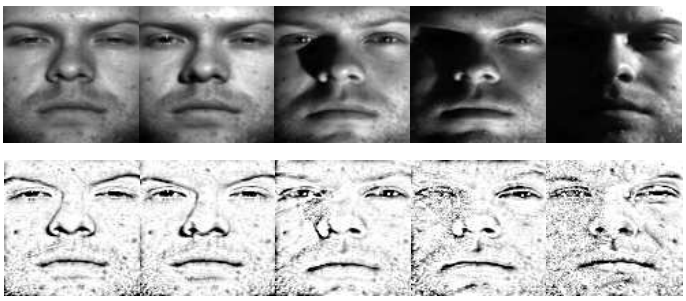

Figure 3.3: Sample images processed with the example code: original images (upper row), ASR processed images - the reflectance functions (lower row)

taking the exponential of the result. A more detailed description of the technique can be found, for example, in [5]. The HOMO technique is implemented in the toolbox with a function that has the following prototype:

`Y=homomorphic(X,boost,CutOff,order,lhistogram_cut,uhistogram_cut)`.

Here, `X` denotes the input grey-scale image to be processed, `boost` stands for the boosting factor of the high frequency components with respect to the low frequency components, `CutOff` denotes the cut-off frequency of the filter (0 - 0.5), `order` denotes the order of the modified Butterworth style filter that is used, and `lhistogram_cut` and `uhistogram_cut` stand for parameters passed to the `histtruncate` function and control the post-processing procedure of the homomorphic filtering based normalization technique. If you type

```
help homomorphic
```

you will get additional information on the function together with several examples of usage.

I would like to add at this point that this function is the work of Dr. Peter Kovesi and would again like to thank him for giving me permission to include it into the INface toolbox.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
Y = Y=normalize8(homomorphic(X,2, .25, 2));
figure,imshow(X);
figure,imshow(uint8(Y));
```

The code reads the image named `sample_image.bmp` into the variable `X`

and applies the HOMO technique with the following parameters to the image: `boost=2`, `CutOff=0.25` and `order=2`. After the execution of the code, both images, i.e., the original one and the processed one, are displayed in two separate figures.

We have applied the above code to several images from the YaleB database. The results of the processing are shown in Fig. 3.3.

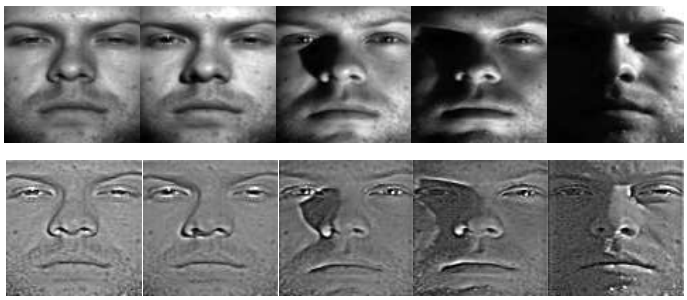

Figure 3.4: Sample images processed with the example code: original images (upper row), HOMO processed images (lower row)

3.1.5 The single scale self quotient image

The single scale self quotient image (SSQ) was introduced to the field of face recognition by Wang et al. in [14]. The technique exhibits similarities to the single scale retinex technique, but unlike the SSR technique uses an anisotropic filter for the smoothing operation. The SSQ technique is implemented in the toolbox with a function that has the following prototype:

```
[R,L]=single_scale_self_quotient_image(X,siz,sigma,normalize);.
```

Here, `X` denotes the input grey-scale image to be processed, `siz` stands for the size of the Gaussian smoothing filter, and `sigma` controls the bandwidth of the filter. The input argument `normalize`$\in \{1, 0\}$ stands for a parameter controlling whether a postprocessing procedure is applied to the normalized image or not. The function returns the "illumination invariant" reflectance `R` and the estimated luminance function `L`. Note here that the luminance function is returned only for visualization purposes, as it is usually only of little value from the perspective of illumination invariant face recognition.

The reader is referred to the original paper for more information on the technique [14]. If you type

```
help single_scale_self_quotient_image
```

you will get additional information on the function together with several examples of usage.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
[R,L]=single_scale_self_quotient_image(X,7,1);
figure,imshow(X);
figure,imshow(R,[]);
figure,imshow(L,[]);
```

The code reads the image named `sample_image.bmp` into the variable `X` and applies the SSQ algorithm with a Gaussian filter of size 7×7 and $\sigma = 1$ to the image in `X`. After the execution of the code, all three images, i.e., the original image, the illumination invariant version (i.e., the reflectance) and the estimated luminance function, are displayed in three separate figures.

We have applied the above code to several images from the YaleB database. The results of the processing are shown in Fig. 3.5.

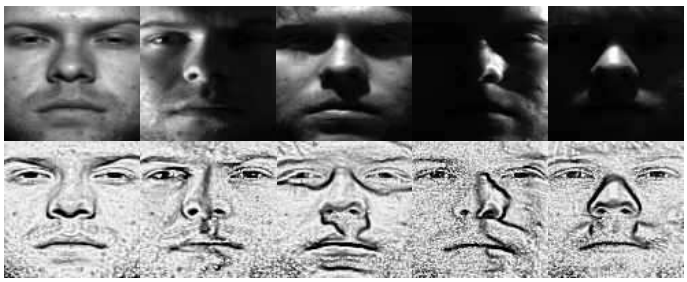

Figure 3.5: Sample images processed with the example code: original images (upper row), SSQ processed images - the reflectance functions (lower row)

3.1.6 The multi scale self quotient image

Like the SSQ technique, the multi scale self quotient image (MSQ) was also introduced to the field of face recognition by Wang et al. in [14]. The technique exhibits similarities to the multi scale retinex technique, but unlike the MSR technique uses an anisotropic filter for the smoothing operation. The MSQ technique is implemented in the toolbox with a function that has the following prototype:

`Y=multi_scale_self_quotient_image(X,siz,sigma,normalize);`.

Here, `X` denotes the input grey-scale image to be processed, `siz` stands for a vector of sizes of the Gaussian smoothing filters, and `sigma` is a vector of same size as `siz` whose values control the bandwidth of the individual filters. The input argument `normalize`$\in \{1, 0\}$ stands for a parameter controlling whether a postprocessing procedure is applied to the normalized image or not. The reader is referred to the original paper for more information on the technique [14]. If you type

```
help multi_scale_self_quotient_image
```

you will get additional information on the function together with several examples of usage.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
Y=multi_scale_self_quotient_image(X,[3 5 11 15],[1 1.1 1.2 1.3]);
figure,imshow(X);
figure,imshow(Y,[]);
```

The code reads the image named `sample_image.bmp` into the variable `X` and applies the MSQ algorithm with Gaussian filter of four scales (defined in `[3 5 11 15]`) and four different bandwidths (defined in `[1 1.1 1.2 1.3]`) to the image in `X`. After the execution of the code, both images, i.e., the original one and the processed one, are displayed in two separate figures.

We have applied the above code to several images from the YaleB database. The results of the processing are shown in Fig. 3.6.

3.1.7 The DCT based normalization technique

The DCT based normalization technique (DCT) was proposed by Chen et al. in [2]. The technique sets a number of DCT coefficients corresponding to low-frequencies to zero and hence tries to achieve illumination invariance. The DCT technique is implemented in the toolbox with a function that has the following prototype:

Figure 3.6: Sample images processed with the example code: original images (upper row), MSQ processed images (lower row)

`Y=DCT_normalization(X,numb,normalize);`.

Here, `X` denotes the input grey-scale image to be processed and `numb` stands for the number of DCT coefficients to set to zero - the coefficients are scanned in a zig-zag manner. The input argument `normalize`$\in \{1, 0\}$ stands for a parameter controlling whether a postprocessing procedure is applied to the normalized image or not. The reader is referred to the original paper for more information on the technique [2]. If you type

```
help DCT_normalization
```

you will get additional information on the function together with several examples of usage.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
Y=DCT_normalization(X,20);
figure,imshow(X);
figure,imshow(Y,[]);
```

The code reads the image named `sample_image.bmp` into the variable `X` and applies the DCT normalization technique to the image in `X`. It sets 20 coefficients to zero. After the execution of the code, both images, i.e., the original one and the processed one, are displayed in two separate figures.

We have applied the above code to several images from the YaleB database. The results of the processing are shown in Fig. 3.7.

Figure 3.7: Sample images processed with the example code: original images (upper row), DCT processed images (lower row)

3.1.8 The wavelet based normalization technique

The wavelet based normalization technique (WA) was proposed by Du and Ward in [3]. The technique applies the discrete wavelet transform to an image and then processes the obtained sub-bands. It emphasizes the matrices of detailed coefficient and applies histogram equalization to the approximate coefficients of the transform. After the manipulation of the individual sub-band the normalized image is reconstructed using the inverse wavelet transform. The WA technique is implemented in the toolbox with a function that has the following prototype:

```
Y=wavelet_normalization(X,fak,wname,mode,normalize);.
```

Here, `X` denotes the input grey-scale image to be processed, `fak` stands for the factor by which the detailed coefficients are scaled, `wname` stands for the wavelet name that is used for the discrete wavelet transform and `mode` denotes a string determining the extension mode of the discrete wavelet transform. The input argument `normalize`$\in \{1, 0\}$ stands for a parameter controlling whether a postprocessing procedure is applied to the normalized image or not. For help on the parameter "mode" please type "help dwtmode" into Matlabs command prompt. The reader is referred to the original paper for more information on the technique [3]. If you type

```
help wavelet_normalization
```

you will get additional information on the function together with several examples of usage.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
Y=wavelet_normalization(X,1.4,'db1');
```

```
figure,imshow(X);
figure,imshow(Y,[]);
```

The code reads the image named `sample_image.bmp` into the variable `X` and applies the WA normalization technique to the image in `X`. Here, it uses a factor of 1.4 to boost the detailed coefficients and employs Daubechies wavelets for the discrete wavelet transform. After the execution of the code, both images, i.e., the original one and the processed one, are displayed in two separate figures.

We have applied the above code to several images from the YaleB database. The results of the processing are shown in Fig. 3.8.

Figure 3.8: Sample images processed with the example code: original images (left), WA processed images (right)

3.1.9 The wavelet denoising based normalization technique

The wavelet denoising based normalization technique (WD) was proposed by Zhang et al. [16]. The technique applies wavelet denoising to an image to obtain an estimate of the luminance and consequently to compute the reflectance. The WD technique is implemented in the toolbox with a function that has the following prototype:

```
[R,L]=wavelet_denoising(X,wname,level, normalize);.
```

Here, `X` denotes the input grey-scale image to be processed, `wname` stands for the wavelet name that is used for the discrete wavelet transform and `level` denotes a scalar value controlling the level of the decomposition. The input argument `normalize`$\in \{1, 0\}$ stands for a parameter controlling whether a postprocessing procedure is applied to the normalized image or not. The reader is referred to the original paper for more information on the technique [16]. The function returns the "illumination invariant" reflectance `R` and the estimated

luminance function `L` (both in the log domain). Note here that the luminance function is returned only for visualization purposes, as it is usually only of little value from the perspective of illumination invariant face recognition.

If you type

```
help wavelet_denoising
```

you will get additional information on the function together with several examples of usage.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
[R,L]=wavelet_denoising(X,'haar');
figure,imshow(X);
figure,imshow(R,[]);
figure,imshow(L,[]);
```

The code reads the image named `sample_image.bmp` into the variable `X` and applies the WD normalization technique to the image in `X` using Haar wavelets and a default value of the decomposition level of 2. After the execution of the code, all three images, i.e., the original one, the processed one (the reflectance) and the estimated luminance, are displayed in three separate figures.

We have applied the above code to several images from the YaleB database. The results of the processing are shown in Fig. 3.9.

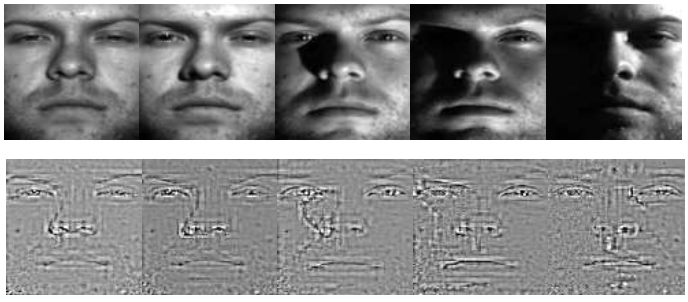

Figure 3.9: Sample images processed with the example code: original images (upper row), WD processed images (lower row) - the reflectance functions

3.1.10 The isotropic diffusion based normalization technique

The isotropic diffusion based normalization technique (IS) uses isotropic smoothing of the image to estimate the luminance function. It represents a simpler variant of the anisotropic diffusion based normalization technique proposed by Gross and Brajovic in [4]. A more detailed description of the technique can be found in [5]. The IS technique is implemented in the toolbox with a function that has the following prototype:

```
[R,L] = isotropic_smoothing(X,param,normalize);.
```

Here, `X` denotes the input grey-scale image to be processed and `param` stands for a scalar value controlling the relative importance of the smoothness constraint. In the papers on diffusion processes this parameter is usually denoted as λ. The input argument `normalize`$\in \{1, 0\}$ stands for a parameter controlling whether a postprocessing procedure is applied to the normalized image or not. The reader is referred to [5] for more information on the technique. The function returns the "illumination invariant" reflectance `R` and the estimated luminance function `L`. Note here that the luminance function is returned only for visualization purposes, as it is usually only of little value from the perspective of illumination invariant face recognition.

If you type

```
help isotropic_smoothing
```

you will get additional information on the function together with several examples of usage.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
Y = isotropic_smoothing(X);
figure,imshow(X);
figure,imshow(Y,[]);
```

The code reads the image named `sample_image.bmp` into the variable `X` and applies the IS normalization technique to the image in `X` using the default value of $\lambda = 7$. After the execution of the code, both images, i.e., the original one and the processed one, are displayed in two separate figures.

We have applied the above code to several images from the YaleB database.

The results of the processing are shown in Fig. 3.10.

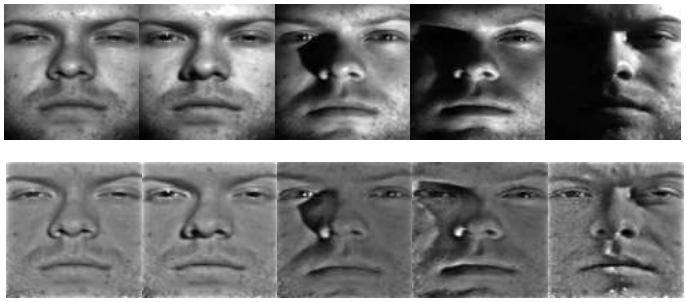

Figure 3.10: Sample images processed with the example code: original images (upper row), IS processed images (lower row) - the reflectance functions

3.1.11 The anisotropic diffusion based normalization technique

The anisotropic diffusion based normalization technique (AS) uses anisotropic smoothing of the image to estimate the luminance function. It was introduced to the field of face recognition by Gross and Brajovic in [4]. The AS technique is implemented in the toolbox with a function that has the following prototype:

```
[R,L] = anisotropic_smoothing(X,param,normalize);.
```

Here, `X` denotes the input grey-scale image to be processed and `param` stands for a scalar value controlling the relative importance of the smoothness constraint. In the papers on diffusion processes this parameter is usually denoted as λ. The input argument `normalize`$\in \{1, 0\}$ stands for a parameter controlling whether a postprocessing procedure is applied to the normalized image or not. The reader is referred to [5] and [4] for more information on the technique. The function returns the "illumination invariant" reflectance `R` and the estimated luminance function `L`. Note here that the luminance function is returned only for visualization purposes, as it is usually only of little value from the perspective of illumination invariant face recognition.

If you type

```
help anisotropic_smoothing
```

you will get additional information on the function together with several examples of usage.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
Y = anisotropic_smoothing(X);
figure,imshow(X);
figure,imshow(Y,[]);
```

The code reads the image named `sample_image.bmp` into the variable `X` and applies the AS normalization technique to the image in `X` using the default value of $\lambda = 7$. After the execution of the code, both images, i.e., the original one and the processed one, are displayed in two separate figures.

We have applied the above code to several images from the YaleB database. The results of the processing are shown in Fig. 3.11.

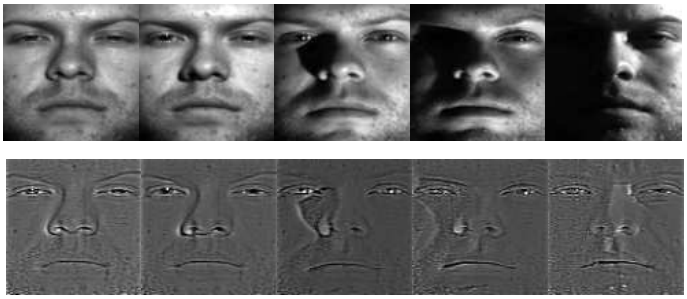

Figure 3.11: Sample images processed with the example code: original images (upper row), AS processed images (lower row) - the reflectance functions

3.1.12 The steerable filter based normalization technique

The steerable filter based normalization technique (SF) uses steerable filters for removing illumination induced appearance variations from the facial images. The SF technique is implemented in the toolbox with a function that has the following prototype:

```
[Y] = steerable_gaussians(X1,sigmas,angles, normalize);.
```

Here, `X` denotes the input grey-scale image to be processed, `sigmas` stands for parameter vector determining the number and bandwidths of the steerable filters and `angles` is a scalar value defining the angular resolution of the filter bank. The input argument `normalize`$\in \{1, 0\}$ stands for a parameter controlling whether a postprocessing procedure is applied to the normalized image or not.

If you type

```
help steerable_gaussians
```

you will get additional information on the function together with several examples of usage.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
Y = steerable_gaussians(X,[0.5,1],6);
figure,imshow(X);
figure,imshow(uint8(Y),[]);
```

The code reads the image named `sample_image.bmp` into the variable `X` and applies the SF normalization technique to the image in `X` using two filter scales with the filters at the first scale having $\sigma = 0.5$ and the filters at the second scale having $\sigma = 1$. On both scales there is a total of 6 filter orientations taken uniformly from the interval $[0, \pi]$. After the execution of the code, both images, i.e., the original one and the processed one, are displayed in two separate figures.

We have applied the above code to sample images from the YaleB database. The results of the processing are shown in Fig. 3.12.

Figure 3.12: Sample images processed with the example code: original imagse (upper row), SF processed images (lower row)

3.1.13 The non-local means based normalization technique

The non-local means based normalization technique (NLM) was proposed by Štruc and Pavešič in [11]. The technique uses the non-local means denoising algorithm to compute the luminance function and consequently to estimate the

reflectance. The NLM technique is implemented in the toolbox with a function that has the following prototype:

```
[R,L] = nl_means_normalization(X,h,N,normalize);.
```

Here, `X` denotes the input grey-scale image to be processed, `h` stands for the parameter controlling the decay of the exponential function used by the technique, and `N` defines the size of the image patches needed. The input argument `normalize`$\in \{1, 0\}$ stands for a parameter controlling whether a postprocessing procedure is applied to the normalized image or not. The reader is referred to [1] for a detailed description of the non-local means algorithm and [11] for more information on the NLM normalization technique. The function returns the "illumination invariant" reflectance `R` and the estimated luminance function `L` (both in the log domain). Note here that the luminance function is returned only for visualization purposes, as it is usually only of little value from the perspective of illumination invariant face recognition.

If you type

```
help nl_means_normalization
```

you will get additional information on the function together with several examples of usage.

I would like to add at this point that this function is based on the non-local means toolbox by Dr. Gabriel Peyré. I would like to thank him for giving me permission to include it into the INface toolbox.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
Y = nl_means_normalization(X);
figure,imshow(X);
figure,imshow(uint8(Y),[]);
```

The code reads the image named `sample_image.bmp` into the variable `X` and applies the NLM normalization technique to the image in `X` using the default values of the technique. After the execution of the code, both images, i.e., the original one and the processed one, are displayed in two separate figures.

We have applied the above code to sample images from the YaleB database. The results of the processing are shown in Fig. 3.13.

Figure 3.13: Sample images processed with the example code: original images (upper row), NLM processed images (lower row) - the reflectance functions

3.1.14 The adaptive non-local means based normalization technique

The adaptive non-local means based normalization technique (ANL) was proposed by Štruc and Pavešič in [11]. The technique uses the adaptive non-local means denoising algorithm to compute the luminance function and consequently to estimate the reflectance. Here, the adaptiveness of the smoothing is a controlled by the images local contrast. The ANL technique is implemented in the toolbox with a function that has the following prototype:

```
[R,L] = adaptive_nl_means_normalization(X,h,N,normalize);.
```

Here, `X` denotes the input grey-scale image to be processed, `h` stands for the parameter controlling the decay of the exponential function (it actually controls the maximum value of the parameter that is linked to the local contrast), and `N` defines the size of the image patches needed. The input argument `normalize`$\in \{1, 0\}$ stands for a parameter controlling whether a postprocessing procedure is applied to the normalized image or not. The reader is referred to [11] for more information on the ANL normalization technique. The function returns the "illumination invariant" reflectance `R` and the estimated luminance function `L` (both in the log domain). Note here that the luminance function is returned only for visualization purposes, as it is usually only of little value from the perspective of illumination invariant face recognition.

If you type

```
help adaptive_nl_means_normalization
```

you will get additional information on the function together with several examples of usage.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
Y = adaptive_nl_means_normalization(X);
figure,imshow(X);
figure,imshow(uint8(Y),[]);
```

The code reads the image named `sample_image.bmp` into the variable `X` and applies the ANL normalization technique to the image in `X` using the default values of the technique. After the execution of the code, both images, i.e., the original one and the processed one, are displayed in two separate figures.

We have applied the above code to sample images from the YaleB database. The results of the processing are shown in Fig. 3.14.

Figure 3.14: Sample images processed with the example code: original images (upper row), ANL processed images (lower row) - the reflectance functions

3.1.15 The modified anisotropic diffusion normalization technique

The modified anisotropic diffusion normalization technique (MAS) represents a modified version of the anisotropic diffusion normalization technique proposed by Gros and Brajovic in [4]. Two modification were introduced into the technique when compared to the original approach: *(i)* the estimate of the local contrast was made more robust by introducing an additional `atan` function. This has the effect of saturating the extreme values that are introduced to the contrast estimate due pixel intensities near 0 in the original face images. *(ii)* a robust postprocessing procedure (proposed in the Tan and Triggs paper) was applied in the final stage of the technique. The MAS technique is implemented in the toolbox with a function that has the following prototype:

```
[R,L] = anisotropic_smoothing_stable(X,param,normalize);.
```

Here, `X` denotes the input grey-scale image to be processed and `param`

stands for a scalar value controlling the relative importance of the smoothness constraint. In the papers on diffusion processes this parameter is usually denoted as λ. The input argument `normalize`$\in \{1, 0\}$ stands for a parameter controlling whether the robust postprocessing procedure is applied to the normalized image or not. The function returns the "illumination invariant" reflectance `R` and the estimated luminance function `L`. Note here that the luminance function is returned only for visualization purposes, as it is usually only of little value from the perspective of illumination invariant face recognition.

If you type

```
help anisotropic_smoothing_stable
```

you will get additional information on the function together with several examples of usage.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
[R,L] = anisotropic_smoothing_stable(X);
figure,imshow(X);
figure,imshow(normalize8(R),[]);
figure,imshow(normalize8(L),[]);
```

The code reads the image named `sample_image.bmp` into the variable `X` and applies the MAS normalization technique to the image in `X` using the default value of $\lambda = 7$. After the execution of the code, all three images, i.e., the original one, the processed one (the reflectance), and the estimated luminance are displayed in three separate figures.

We have applied the above code to several images from the YaleB database. The results of the processing are shown in Fig. 3.15.

3.1.16 The Gradientfaces normalization technique

The Gradientfaces-based normalization technique (GRF) represents a normalization technique first proposed in [17]. The function computes the orientation of the image gradients in each pixel of the face images and uses the computed face representation as an illumination invariant version of the input image. The GRF technique is implemented in the toolbox with a function that has the following prototype:

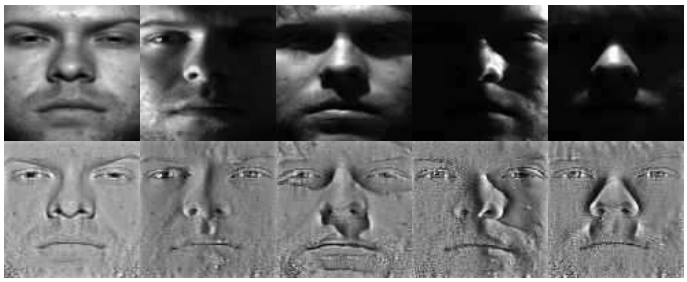

Figure 3.15: Sample images processed with the example code: original images (upper row), MAS processed images (lower row) - the reflectance functions

```
Y = gradientfaces(X,sigma, normalize);.
```

Here, `X` denotes the input grey-scale image to be processed and `sigma` stands for the standard deviation of the Gaussian derivative filters used for estimating the image gradient. The input argument `normalize`$\in \{1, 0\}$ stands for a parameter controlling whether a postprocessing procedure is applied to the normalized image or not. The reader is referred to [17] for more information on the GRF normalization technique.

If you type

```
help gradientfaces
```

you will get additional information on the function together with several examples of usage.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
Y = gradientfaces(X);
figure,imshow(X);
figure,imshow(normalize8(Y),[]);
```

The code reads the image named `sample_image.bmp` into the variable `X` and applies the GRF normalization technique to the image in `X` using the default value of `sigma=0.75`. After the execution of the code, both, i.e., the original one and the processed one, are displayed in two separate figures.

We have applied the above code to several images from the YaleB database. The results of the processing are shown in Fig. 3.16.

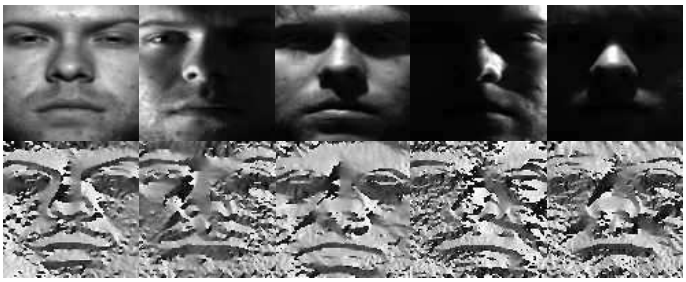

Figure 3.16: Sample images processed with the example code: original images (upper row), GRF processed images (lower row)

3.1.17 The single scale Weberfaces normalization technique

The single scale Weberfaces normalization technique (WEB) represents a normalization technique first proposed in [13]. The function computes the relative gradient in the form of a modified Weber contrast and uses the computed face representation as an illumination invariant version of the input image. The WEB technique is implemented in the toolbox with a function that has the following prototype:

```
Y = weberfaces(X,sigma, nn, alfa, normalize);.
```

Here, `X` denotes the input grey-scale image to be processed, `sigma` stands for the standard deviation of the Gaussian filter that is applied in the first step of the technique, `nn` denotes the size of the local neighborhood over which the contrast (or to use the notaion of the authors the relative gradient) is computed, e.g., 9, 25, ..., $(2n+1)^2; n \in \mathcal{N}^+$, `alfa` is the magnification parameter used for balancing the input range of the `atan` function and `normalize` $\in \{1, 0\}$ is the parameter that controls whether the input result is scaled to the 8-bit interval or not. The reader is referred to [13] for more information on the WEB normalization technique.

If you type

```
help weberfaces
```

you will get additional information on the function together with several examples of usage.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
Y = weberfaces(X);
```

```
figure,imshow(X);
figure,imshow(normalize8(Y),[]);
```

The code reads the image named `sample_image.bmp` into the variable `X` and applies the WEB normalization technique to the image in `X` using the default values of `sigma=0.75`, `nn = 9`, `alfa = 2`, and `normalize = 1`. After the execution of the code, both, i.e., the original one and the processed one, are displayed in two separate figures.

We have applied the above code to several images from the YaleB database. The results of the processing are shown in Fig. 3.17.

Figure 3.17: Sample images processed with the example code: original images (upper row), WEB processed images (lower row)

3.1.18 The multi scale Weberfaces normalization technique

The multi scale Weberfaces normalization technique (MSW) is a straight forward extension of the single scale Weberfaces approach proposed in [13]. The function computes the relative gradient in the form of a modified Weber contrast for different neighborhood sizes and uses a linear combination of the computed face representations as an illumination invariant version of the input image. The MSW technique is implemented in the toolbox with a function that has the following prototype:

```
Y = multi_scale_weberfaces(X,sigma, nn, alfa, weights, normalize);.
```

Here, `X` denotes the input grey-scale image to be processed, `sigma` stands for a vector of standard deviations of the Gaussian filters that are applied in the first step of the technique, `nn` denotes a vector of sizes of the local neighborhoods over which the contrasts (or to use the notaion of the authors the relative gradients) are computed, e.g., [3 5 7], `alfa` is a vector of magnification

parameters used for balancing the input range of the `atan` function, `weights` is a vector of weight used for combining the individual Weberfaces and `normalize` $\in \{1, 0\}$ is the parameter that controls whether the input result is scaled to the 8-bit interval or not. Note that all input vectors (i.e., `sigma, nn, alfa, weights`) need to have the same number of elements.

If you type

```
help multi_scale_weberfaces
```

you will get additional information on the function together with several examples of usage.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
Y = multi_scale_weberfaces(X);
figure,imshow(X);
figure,imshow(normalize8(Y),[]);
```

The code reads the image named `sample_image.bmp` into the variable `X` and applies the MSW normalization technique to the image in `X` using the default values of `sigma=[1 0.75 0.5]`, `nn = [9 25 49]`, `alfa = [2 0.2 0.02]`, `weights = [1 1 1]` and `normalize = 1`. After the execution of the code, both, i.e., the original one and the processed one, are displayed in two separate figures.

We have applied the above code to several images from the YaleB database. The results of the processing are shown in Fig. 3.18.

Figure 3.18: Sample images processed with the example code: original images (upper row), MSW processed images (lower row)

3.1.19 The large- and small-scale features normalization technique

The large- and small-scale features normalization technique (LSSF) is a normalization technique first proposed in [15]. The technique normalizes the input image by first computing the reflectance and luminance functions of the image and then further processing both computed functions using a second round of normalization. The technique implemented in the toolbox uses the SSR technique in both steps, but does not implement the non-point light technique which requires training data, since this would limit the applicability of the technique to frontal images. The SSR technique is implemented in the toolbox with a function that has the following prototype:

```
Y = lssf_norm(X, normalize);.
```

Here, `X` denotes the input grey-scale image to be processed and `normalize` $\in \{1, 0\}$ is the parameter that controls whether the input result is scaled to the 8-bit interval or not. The reader is referred to [15] for more information on the LSSF normalization technique.

If you type

```
help lssf_norm
```

you will get additional information on the function together with several examples of usage.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
Y = lssf_norm(X);
figure,imshow(X);
figure,imshow(normalize8(Y),[]);
```

The code reads the image named `sample_image.bmp` into the variable `X` and applies the LSSF normalization technique to the image in `X` using the default value `normalize = 1` After the execution of the code, both, i.e., the original one and the processed one, are displayed in two separate figures.

We have applied the above code to several images from the YaleB database. The results of the processing are shown in Fig. 3.19.

Figure 3.19: Sample images processed with the example code: original images (upper row), LSSF processed images (lower row)

3.1.20 The Tan and Triggs normalization technique

The Tan and Triggs normalization technique (TT) as the name suggests is a normalization technique first proposed by Tan and Triggs in [10]. The technique normalizes the input image through the use of a processing chain that first applies gamma correction to the input image, then subjects the corrected image to DoG filtering and finally employs a robust post-processor to produce the final result. The TT technique is implemented in the toolbox with a function that has the following prototype:

```
Y = tantriggs(X,gamma, normalize);.
```

Here, `X` denotes the input grey-scale image to be processed, `gamma` stands for the gamma parameter of the gamma correction and `normalize` $\in \{1, 0\}$ is the parameter that controls whether the input result is scaled to the 8-bit interval or not. The reader is referred to [10] for more information on the TT normalization technique.

If you type

```
help lssf_norm
```

you will get additional information on the function together with several examples of usage.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
Y = tantriggs(X);
figure,imshow(X);
figure,imshow(normalize8(Y),[]);
```

The code reads the image named `sample_image.bmp` into the variable `X` and applies the TT normalization technique to the image in `X` using the default values `gamma = 0.2` and `normalize = 1`. After the execution of the code, both, i.e., the original one and the processed one, are displayed in two separate figures.

We have applied the above code to several images from the YaleB database. The results of the processing are shown in Fig. 3.20.

Figure 3.20: Sample images processed with the example code: original images (upper row), TT processed images (lower row)

3.1.21 The DoG filtering-based normalization technique

The DoG filtering-based normalization technique (DOG) is a normalization technique which relies on the difference of Gaussians filter to produce the normalized image. Basically it applies a bandpass filter to the input image and produces a normalized version of it. Note that before you use the filter you have to apply gamma correction or the log transform to the image or the result will not be what you were hoping for. The DOG technique is implemented in the toolbox with a function that has the following prototype:

```
Y = dog(X,sigma1, sigma2, normalize);.
```

Here, `X` denotes the input grey-scale image to be processed, `sigma1` stands for the standard deviation of the high-pass Gaussian filter, `sigma2` stands for the standard deviation of the low-pass Gaussian filter and `normalize` $\in \{1, 0\}$ is the parameter that controls whether the input result is postprocessed or not.

If you type

```
help dog
```

you will get additional information on the function together with several

examples of usage.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
Y = dog(log(X+1));
figure,imshow(X);
figure,imshow(normalize8(Y),[]);
```

The code reads the image named `sample_image.bmp` into the variable `X` and applies the DOG normalization technique to the image in `X` using the default values `sigma1 = 1`, `sigma2=2` and `normalize = 1`. After the execution of the code, both, i.e., the original one and the processed one, are displayed in two separate figures.

We have applied the above code to several images from the YaleB database. The results of the processing are shown in Fig. 3.21.

Figure 3.21: Sample images processed with the example code: original images (upper row), DOG processed images (lower row)

3.2 The *postprocessors* folder

The folder named *postprocessors* contains the implementations of two postprocessing techniques. Specifically, it contains the following functions:

- `histtruncate`, and
- `robust_postprocessor`,

These functions were included in the toolbox to alow the evaluation of the effect of applying different postprocessing techniques to the photometrically normalized images. Note that in the INFace toolbox version 1.0 a default postprocessing procedure was applied after all photometric normalization techniques,

namely, truncation of the histogram ends and scaling of the images dynamic range to the 8-bit interval. With version 2.0 we made the postprocessing optional. The procedure can be skipped by setting the `normalize` parameter, which is now featured in all function prototypes, to 0. However, we do not encourage setting the parameter to zero, since postprocessing is often crucial for the performance of the photometric normalization technique. If you do not plan to use your own postprocessor or to replace the default postprocessing procedure with another one from this toolbox, leave the value of the parameter `normalize` set to its default value of 1.

3.2.1 The `histtruncate` function

The `histtruncate` function truncates a specified percentage of the lower and upper ends of an image histogram. The function has the following prototype:

`[Y, sortv] = histtruncate(X, lower, upper)`.

Here, `X` denotes the input grey-scale image to be processed, `lower` denotes the percentage of the lower part and `upper` denotes the percentage of the upper part of the histogram that gets truncated. If you type

```
help histtruncate
```

you will get additional information on the function together with several examples of usage.

As noted in the functions header, this function was again provided by Dr. Peter Kovesi.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
Y = histtruncate(X, 3, 5);
figure,imshow(X);
figure,hist(X(:),255);
figure,imshow(uint8(Y),[]);
figure,hist(normalize8(Y(:)),255);
```

The code reads the image named `sample_image.bmp` into the variable `X` and truncates the histogram of `X` using the selected percentage of the lower and

upper part of the histogram. After the execution of the code the original and the processed image together with their histograms, are displayed in four separate figures.

We have applied the above code to a sample image from the YaleB database. The results of the processing are shown in Fig. 3.22.

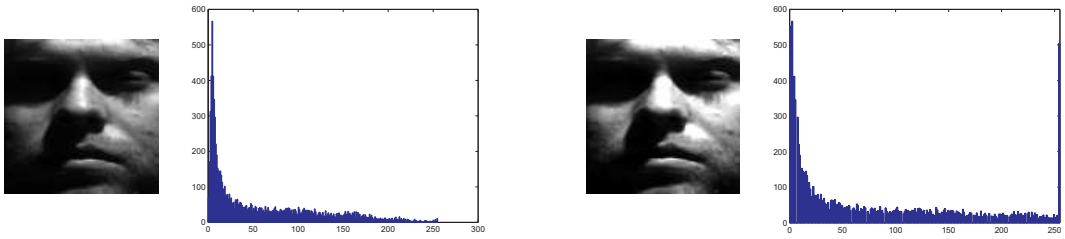

Figure 3.22: A sample image processed with the example code: original image with its histogram (left), processed image with its histogram (right)

Below is an example of the effect of histogram truncation on photometrically normalized images (with the SSR technique).

Figure 3.23: Sample images from the YaleB database normalized with the SSR technique: without truncation (upper row), after histogram truncation (lower row)

3.2.2 The `robust_postprocessor` function

The `robust_postprocessor` function applies to the robust postprocessing procedure from the Tan and Triggs paper [10] to the input image. The function has the following prototype:

```
Y=robust_postprocessor(X, alfa, tao);.
```

Here, `X` denotes the input grey-scale image to be processed, and `alfa` and `tao` denote hyper parameters of the technique. If you type

```
help robust_postprocessor
```

you will get additional information on the function together with several examples of usage.

We have applied the above postprocessing procedure to sample images from the YaleB database that were photometrically normalized using the Tan and Triggs method (i.e., using only the gamma correction and Dog filtering parts). Fig. 3.24 show the effect the postprocessing procedure has on the result. As we can see, it is crucial to ensure comparable appearance of the images.

Figure 3.24: Sample image from the YaleB database processed with the Tan and Triggs method: without robust postprocessing (upper row), with robust postprocessing (lower row).

3.3 The *histograms* folder

The folder named *histograms* contains the implementations of a number of histogram manipulating techniques. Specifically, it contains the following functions:

- `rank_normalization`, and
- `fitt_distribution`,

These functions were included in the toolbox, as they are able to manipulate the histograms of the facial images, which is a common pre- or post-processing step to photometric normalization. In fact, several studies have shown that histogram equalization or histogram remapping in conjunction with photometric normalization techniques results in better face recognition performance than using photometric normalization techniques on their own. In the remainder of this section we will focus on the description of the two techniques contained in the *histograms* folder.

3.3.1 The `rank_normalization` function

The `rank_normalization` function applies rank normalization to the pixel intensity values of an image. This means that all pixels in an image are ordered from the most negative to the most positive (from the one with the smallest intensity value to the one with the largest intensity valuel). After the ordering the first pixel is assigned a rank of one, the second the rank of two, ..., and the last is assigned a rank of N, where N is the number of pixels in the image. This procedure is identical to histogram equalization, except for the interval to which the intensity values are mapped to. (For options on the interval take a look at the description of the parameter "mode"). Unlike Matlabs internal function "histeq" this function also works with doubles, works faster and provides more flexibility regarding the output range of the pixel intensity values. The function has the following prototype:

`Y=rank_normalization(X,mode,updown)`.

Here, `X` denotes the input grey-scale image to be processed, `mode` is a string determining the range of the output intensity values and `updown` a string controlling the sort operation performed by the function. If you type

```
help rank_normalization
```

you will get additional information on the function together with several examples of usage.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
Y=rank_normalization(X,'two');
figure,imshow(X);
figure,hist(X(:),255);
figure,imshow(uint8(Y),[]);
figure,hist(normalize8(Y(:)),255);
```

The code reads the image named `sample_image.bmp` into the variable `X` and performs rank normalization on `X`. After the execution of the code the original and the processed image together with their histograms, are displayed in four separate figures.

We have applied the above code to a sample image from the YaleB database. The results of the processing are shown in Fig. 3.25.

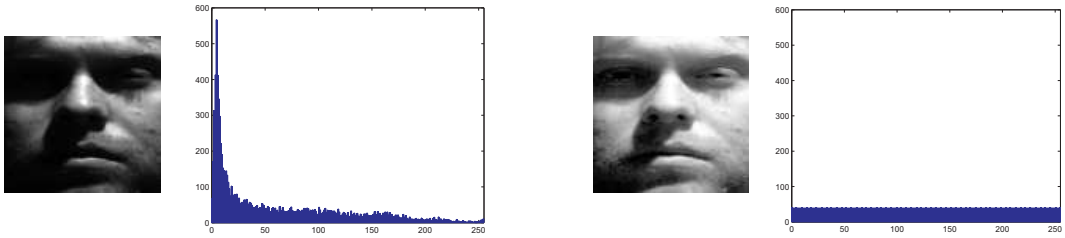

Figure 3.25: A sample image processed with the example code: original image with its histogram (left), processed image with its histogram (right)

3.3.2 The `fitt_distribution` function

The function `fitt_distribution` fits a predefined distribution to the histogram of an image. The function supports three target distributions, namely, the normal, the lognormal and the exponential distribution. The function has the following prototype:

```
Y=fitt_distribution(X,distr,param).
```

Here, `X` denotes the input grey-scale image to be processed, `distr` is a scalar value (i.e, 1,2,3) and determines the target distribution and `param` is either a vector of parameters or a single parameter depending on the target distribution. If the target distribution is normal or lognormal then `param` has to be a 1×2 vector containing the mean and standard deviation of the distribution. If the target distribution is exponential the only parameter is λ. If you type

```
help fitt_distribution
```

you will get additional information on the function together with several examples of usage. The reader is referred to [12] for more information on histogram remapping.

An example of the use of the function is shown below:

```
X=imread('sample_image.bmp');
Y=fitt_distribution(X,1,[0,1]);
figure,imshow(X);
```

```
figure,hist(X(:),255);
figure,imshow(uint8(Y),[]);
figure,hist(normalize8(Y(:)),255);
```

The code reads the image named `sample_image.bmp` into the variable `X` and fits the normal distribution with a mean value of 0 and a standard deviation of 1 to the histogram of `X`. After the execution of the code the original and the processed image together with their histograms, are displayed in four separate figures.

We have applied the above code to a sample image from the YaleB database. The results of the processing are shown in Fig. 3.26.

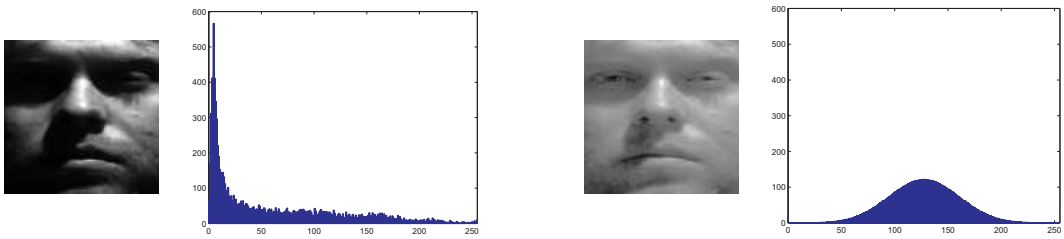

Figure 3.26: A sample image processed with the example code: original image with its histogram (left), processed image with its histogram (right)

3.4 The *auxilary* folder

The folder named *auxilary* contains functions needed for the the photometric normalization techniques to work. Specifically, the folder contains the following functions:

- `gamma_correction`,
- `threshold_filtering`,
- `adjust_range`,
- `compute_patch_library`,
- `highboostfilter`,
- `highpassfilter`,

- `lowpassfilter`,
- `normalize8`,
- `pca`,
- `perform_nl_means`,
- `perform_lowdim_embedding`,
- `perform_nl_means_adap`, and
- `symmetric_extension`.

Even though some of these function could be used on their own, they are included in the toolbox only to provide some functionality to the main functions in the *photometric* and *histograms* folders. If someone is interested in their functionality, he/she can access the help of the function of interest by typing:

```
help function_name
```

or for a list of the basic functionality of all functions in the *auxilary* folder:

```
help auxilary
```

3.5 The *mex* folder

The *mex* folder contains only a couple of C/C++ files needed for the creation of the MEX files employed by the non-local means and adaptive non-local means algorithm. The code in `perform_nlmeans_mex.cpp` was provided by Dr. Gabriel Peyré.

3.6 The *other* folder

The *other* folder contains several BibTex files with references to papers describing the individual techniques included in the INface toolbox v2.0.

3.7 The *demo* folder

The *demo* folder contains three scripts named:

- `photometric_demo`,
- `histograms_demo`,
- `combin_demo`,
- `make_new_method_demo`,
- `luminance_demo`,

The first is a demo script demonstrating the deployment of all photometric normalization techniques on a sample image, the second is a demo script demonstrating the deployment of the histogram manipulation functions, the third is a demo script demonstrating how to combine photometric normalization techniques and one of the histogram manipulation function, the fourth is a script demonstrating how different functions from the toolbox can be combined to create new normalization techniques, and the fifth is a demo script demonstrating how the luminance functions can be computed.

4. Using the Help

The toolbox contains some basic help which offers additional information on each of the functions and scripts in the INface toolbox. The help in the functions is intended to be used as supplementary information on the functionality of the toolbox.

4.1 Toolbox and folder help

The most basic information on the toolbox can be accessed by typing:

`help INface_tool`,

or if you have changed the name of the folder:

`help new_folder_name`.

This command displays a list of folders in the toolbox and a basic description of the functionality of each function and/or script in these folders.

You can also access only individual folder help by typing, e.g., for the *photometric* folder:

`help photometric`.

4.2 Function help

To access the help of the individual functions (like always) just call the `help` command followed by the name of the function of interest:

`help function_name`.

All functions (and scripts) are equipped with an extensive help section describing the functionality of the function and also include a reference to the paper, where the implemented technique was proposed.

5. The INFace homepage

Together with version v2.0 of the INFace toolbox we have also created an official toolbox homepage. The page is available from:

`http://luks.fe.uni-lj.si/sl/osebje/vitomir/face_tools/INFace/index.html`

However, since we are currently in the process of redesigning our laboratory web page, the final URL of the web site may still change.

Figure 5.1: Screenshot of the official website

Note that only the official website features the up to date version of the

toolbox with all bug fixes and most recent changes, while only major releases are distributed to other repositories as well. Hence, if you have obtained the toolbox from any other location that the official website, I suggest that you make sure that you have obtained the most recent version.

6. Change Log

There has been a significant change in the INFace toolbox from version 1.0 to version 2.0. Several new photometric techniques were added, some bugs were fixed, novel pre- and post-processing techniques were included and much more ...

A brief list of changes that were incorporated to the INFace toolbox version 2.0:

- gamma correction was added (auxilary function),
- range adjustment of the image intensities was added (auxilary function),
- threshold filtering was added (auxilary function),
- DoG filtering was added (photometric normalization),
- Tan and Triggs technique was added (photometric normalization),
- robust Tan and Triggs post-processing was added (postprocessing function),
- Weberfaces technique was added (photometric normalization),
- multi-scale Weberfaces technique was added (photometric normalization),
- modified version of anisotropic diffusion technique was added (photometric normalization),
- DCT-based normalization technique was fixed (bug fix),
- option for retrieving luminance functions was added to some functions (general update),
- default post-processing (i.e., histogram truncation) technique was made optional (general update),
- added new demo scripts (general update),
- return values in case of error were corrected (general update),

- all of the help sections (i.e., the function headers) were updated (general update),
- an install validation script was added (general update),
- several BibTex files were added to the "other" folder (general update),
- a postprocessing folder was added to the toolbox hierarchy (general update),
- the a new user manual was written (general update), and
- a new toolbox homepage was created (new webpage).

Note that one of the changes made to the toolbox was the addition of an additional input argument to all photometric normalization techniques. The last argument of all the photometric techniques is now the parameter `normalize`, which controls whether the default postprocessing procedure is applied to the normalized image or not. This option was added to enable researchers to experiment with their own postprocessors. However, if you do not intend to use your own postprocessor, you should not alter the default value of this argument, since postprocessing is usually one of the most important steps of the normalization procedure.

The presented addition allows for the development of illumination normalization techniques comprised of:

- a preprocessing technique,
- the main photometric normalization technique, and
- a postprocessing technique.

7. Conclusion

The current version of the toolbox (INface v2.0) includes Matlab functions that implement 21 photometric normalization techniques from the literature. If you have authored a novel normalization technique, you are welcome to send me the paper of the technique (or optionally the source code) and I will try to include the technique in future versions of the toolbox. You can find my contact information by following this link:

`http://luks.fe.uni-lj.si/en/staff/vitomir/index.html`.

Thank you for taking an interest in the INface toolbox.

References

[1] A. Buades, J.M Morel B. Coll. Multiscale modeling and simulation (siam interdisciplinary journal). *Pattern Recognition*, Vol. 4, No. 2, str. 490–530, 2005.

[2] W. Chen, M.J. Er, S. Wu. Illumination compensation and normalization for robust face recognition using discrete cosine transform in logarithmic domain. *IEEE Transactions on Systems, Man and Cybernetics - part B*, Vol. 36, No. 2, str. 458–466.

[3] S. Du, R. Ward. Wavelet-based illumination normalization for face recognition. *Proc. of the IEEE International Conference on Image Processing*, September.

[4] R. Gross, V. Brajovic. An image preprocessing algorithm for illumination invariant face recognition. *Proc. of the 4th International Conference on Audio- and Video-Based Biometric Personal Authentication*, July.

[5] G. Heusch, F. Cardinaux, S. Marcel. Lighting normalization algorithms for face verification. *IDIAP-com 05-03*, , , March.

[6] D.J. Jobson, Z. Rahman, G.A. Woodell. A multiscale retinex for bridging the gap between color images and the human observations of scenes. *IEEE Transactions on Image Processing*, Vol. 6, No. 7, str. 965–976, 1997.

[7] D.J. Jobson, Z. Rahman, G.A. Woodell. Properties and performance of a center/surround retinex. *IEEE Transactions on Image Processing*, Vol. 6, No. 3, str. 451–462, 1997.

[8] E.H. Land, J.J. McCann. Lightness and retinex theory. *Journal of the Optical Society of America*, Vol. 61, No. 1, str. 1–11, 1971.

[9] Y.K. Park, S.L. Park, J.K. Kim. Retinex method based on adaptive smoothing for illumination invariant face recognition. *Signal Processing*, Vol. 88, No. 8, str. 1929–1945, 2008.

[10] X. Tan, B. Triggs. Enhanced local texture sets for face recognition under difficult lighting conditions. *IEEE Transactions on Image Processing*, Vol. 19, No. 6, str. 1635–1650, 2010.

[11] V. Štruc, N. Pavešić. Illumination invariant face recognition by non-local smoothingn. *Proceedings of the BIOID Multicomm*, September.

[12] V. Štruc, J. Žibert, N. Pavešić. Histogram remapping as a preprocessing step for robust face recognition. *WSEAS Transactions on Information Science and Applications*, Vol. 6, No. 3, str. 520–529, 2009.

[13] B. Wang, W. Li, W. Yang, Q. Liao. Illumination normalization based on weber's law with application to face recognition. *IEEE Signal Processing Letters*, Vol. 18, No. 8, str. 462–465, 2011.

[14] H. Wang, S.Z. Li, Y. Wang, J. Zhang. Self quotient image for face recognition. *Proceedings of the International Conference on Pattern Recognition*, October.

[15] X. Xie, W.S. Zheng, J. Lai, P.C. Yuen, C.Y. Suen. Normalization of face illumination based on large- and small- scale features. *IEEE Transactions on Image Processing*, Vol. 20, No. 7, str. 1807–1821, 2011.

[16] T. Zhang, B. Fang, Y. Yuan, Y.Y. Tang, Z. Shang, D. Li, F. Lang. Multiscale facial structure representation for face recognition under varying illumination. *Pattern Recognition*, Vol. 42, No. 2, str. 252–258.

[17] T. Zhang, Y.Y. Tang, B. Fang, Z. Shang, X. Liu. Face recognition under varying illumination usng gradientfaces. *IEEE Transactions on Image Processing*, Vol. 18, No. 11, str. 2599–2606, 2009.