University of Ljubljana

Faculty of Electrotechnical Engineering

Vitomir Štruc

The PhD face recognition toolbox

Toolbox description and user manual

Ljubljana, 2012

Thanks

I would like to thank Alpineon Ltd and the Slovenian Research Agency (ARRS) for funding the postdoctoral project BAMBI (Biometric face recognition in ambient intelligence environments, Z2-4214), which made the PhD face recognition toolbox possible.

Foreword

The PhD (**P**retty **h**elpful **D**evelopment functions for) face recognition toolbox 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 PhD face recognition toolbox includes implementations of some of the most popular face recognition techniques, such as Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), Kernel Principal Component Analysis (KPCA), Kernel Fisher Analysis (KFA). It features functions for Gabor filter construction, Gabor filtering, and all other tools necessary for building Gabor-based face recognition techniques.

In addition to the listed techniques there are also a number of evaluation tools available in the toolbox, which make it easy to construct performance curves and performance metrics for the face recognition you are currently assessing. These tools allow you to compute ROC (Receiver Operating Characteristics) curves, EPC (Expected performance curves) curves, and CMC (cumulative match score curves) curves.

Most importantly (especially for beginners in this field), the toolbox also contains several demo scripts that demonstrate how to build and evaluate a complete face recognition system. The demo scripts show how to align the face images, how to extract features from the aligned, cropped and normalized images, how to classify these features and finally how to evaluate the performance of the complete system and present the results in the form of performance curves and corresponding performance metrics.

This document describes the basics of the toolbox, from installation to usage. It contains some simple examples of the usage of each function and the corresponding results. However, for more information on the theoretical background of the functions, the reader is referred to papers from the literature, which is vast. Hence, it shouldn't be to difficult to find the information you are looking for.

As a final word, I would like to say that I have written this document really fast and have not had the time to proof read it. So please excuse any typos or awkward formulations you may find.

Contents

1. Installing the toolbox

The PhD 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 *PhD_tool* should appear and in this folder seven additional directories should be present, namely, *utils*, *plots*, *features*, *eval*, *demos*, *classification* and *bibs*. 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 *PhD_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_PhD
```

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. The installation script was tested with Matlab version 7.11.0.584 (R2010b) on a 64-bit Windows 7 installation.

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. 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 PhD toolbox features a script, which aims 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_PhD_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. Several warnings may be printed to the command window during the validation process. This is simply a consequence of dummy data that is being used for the validation. The warnings do not indicate any errors in the installation process and can be ignored. If the installation completed successfully, you should see a report similar to the one below:

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

VALIDATION REPORT:

Function ``register_face_based_on_eyes'' is working properly.
Function ``construct_Gabor_filters_PhD'' is working properly.
Function ``compute_kernel_matrix_PhD'' is working properly.
Function ``resize_pc_comps_PhD'' is working properly.
Function ``plot_ROC_PhD'' is working properly.
Function ``plot_EPC_PhD'' is working properly.
Function ``plot_CMC_PhD'' is working properly.
Function ``filter_image_with_Gabor_bank_PhD'' is working properly.
Function ``linear_subspace_projection_PhD'' is working properly.
Function ``nonlinear_subspace_projection_PhD'' is working properly.
Function ``produce_phase_congruency_PhD'' is working properly.
Function ``perform_pca_PhD'' is working properly.
Function ``perform_lda_PhD'' is working properly.
Function ``perform_kpca_PhD'' is working properly.
Function ``perform_kfa_PhD'' is working properly.
Function ``produce_ROC_PhD'' is working properly.
Function ``produce_EPC_PhD'' is working properly.
Function ``produce_CMC_PhD'' is working properly.
Function ``evaluate_results_PhD'' is working properly.
Function ``return_distance_PhD'' is working properly.
Function ``nn_classification_PhD' 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.

Note that the `check_PhD_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, "The Complete Gabor-Fisher Classifier for Robust Face Recognition", EURASIP Advances in Signal Processing, pp. 26, 2010, doi=10.1155/2010/847680.

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 *bibs* 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.

The PhD (**P**retty **h**elpful **D**evelopment functions for) face recognition toolbox in its current form is a collection of useful functions that implement:

- four subspace projection techniques:
  - principal component analysis (PCA) - linear technique,
  - linear discriminant analysis (LDA) - linear technique,
  - kernel principal component analysis (KPCA) - non-linear technique,
  - kernel Fisher analysis (KFA) - non-linear technique,
- Gabor filter construction and Gabor filtering techniques,
- phase congruency based feature extraction techniques,
- techniques for constructing three types of performance graphs:
  - Expected performance curves (EPC),
  - Receiver Operating Characteristics (ROC) curves,
  - Cumulative Match Score curves (CMC),
- face registration techniques (based on eye coordinates),
- nearest neighbor based classification techniques, and
- several demo scripts.

The toolbox contains seven folders, named, *utils*, *plots*, *features*, *eval*, *classification*, *demos* and *bibs*. In the remainder of this chapter we will focus on the description of the contents of each of these folders.

3.1 The *utils* folder

The folder named *utils* contains utility (auxilary) functions that in most cases are intended to be used on their own. Instead they provide some functionality to higher level functions of the toolbox. The following functions are featured in the *utils* folder:

- `compute_kernel_matrix_PhD`,
- `construct_Gabor_filters_PhD`,
- `register_face_based_on_eyes`, and
- `resize_pc_comps_PhD`.

A basic description of the functionality of the above listed functions is given in the remainder of this document.

3.1.1 The `compute_kernel_matrix_PhD` function

The `compute_kernel_matrix_PhD` function is an utility function used by the non-linear subspace projection techniques KPCA and KFA. It constructs kernel matrices from the input data using the specified kernel function. The function has the following prototype:

```
kermat = compute_kernel_matrix_PhD(X,Y,kernel_type,kernel_args).
```

Here, `X` and `Y` denote the input data based on which the kernel matrix `kermat` is computed. `kernel_type` denotes the type of the kernel function to use when computing the kernel matrix[1], and `kernel_args` stands for the corresponding arguments of the selected kernel function. If you type

```
help compute_kernel_matrix_PhD
```

[1]Currently only polynomial, fractional power polynomial, and sigmoidal kernel functions are supported.

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:

```
kermat = compute_kernel_matrix_PhD(X,Y,'poly',[1 2]);;
```

The example code computes a kernel matrix based on the input data matrices `X` and `Y` using a polynomial function of the following form:

$$\texttt{kermat} = (X^T Y + 1)^2, \tag{3.1}$$

where 1 and 2 were provided as input arguments. As a result the code returns the computed kernel matrix `kermat`.

3.1.2 The `construct_Gabor_filters_PhD` function

The `construct_Gabor_filters_PhD` function is an auxilary function that can be used to construct a filter bank of complex Gabor filters. The function can be called with only a small number of input arguments, in which case the default values for the filter bank parameters are used for the construction. These default parameters correspond to the most common parameters used in conjunction with face images of size 128×128 pixels. Optionally, you can provide your own filter bank parameters. The complete function prototype is:

```
fbank = construct_Gabor_filters_PhD(norie, nscale, size, fmax, ni,
gamma, separation);.
```

The function returns a structure `fbank`, which contains the spatial and frequency representations of the constructed Gabor filter bank as well as some meta-data. The input arguments of the function represent filter bank parameters, which are in detail described in the help section of the function. More information on the parameters can also be found in the literature, e.g., [11].

If you type

```
help construct_Gabor_filters_PhD
```

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:

```
filter_bank = construct_Gabor_filters_PhD(8, 5, [128 128]);
figure
imshow(real(filter_bank.spatial3,4(64:64+128,64:64+128)),[])
title('An example of the real part of a Gabor filter.')
figure
imshow(abs(filter_bank.spatial3,4(64:64+128,64:64+128)),[])
title('An example of the magnitude of a Gabor filter.')
```

The code constructs a Gabor filter bank comprising filters of 8 orientations and 5 scales, which is intended to be used with facial images of size 128×128 pixels. It then presents an example of the real part and an example of the magnitude of one of the filters from the bank.

When running the above sample code you should get an result that looks like this:

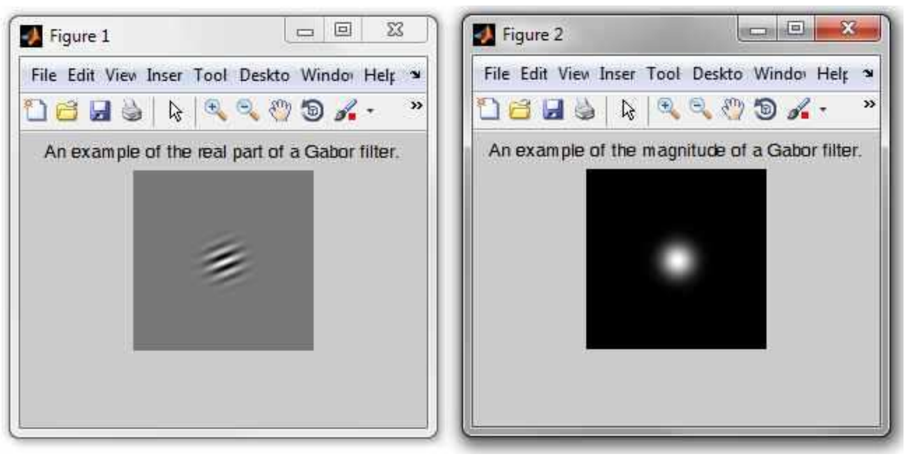


Figure 1
File Edit View Insert Tools Desktop Window Help
An example of the real part of a Gabor filter.
Figure 2
File Edit View Insert Tools Desktop Window Help
An example of the magnitude of a Gabor filter.

Figure 3.1: The result of the above code

3.1.3 The `register_face_based_on_eyes` function

The `register_face_based_on_eyes` function is one of the few functions from the *utils* that is actually meant to be used on its own. The function extracts the facial region from an image given the eye center coordinates and normalizes the image in terms of geometry and size. Thus, it rotates the image in such a way, that the face is upright, then crops the facial region and finally rescales it to a standard size. If the input image is a color image, the extracted region is also an color image (the same applies to grey-scale images). The function has the following prototype:

`Y` = register_face_based_on_eyes(X,eyes,size1).

Here, `X` denotes the input image, from which to extract the face, `eyes` denotes a structure with eye coordinates, and `size1` denotes a parameter that controls to which size the images is rescaled in the final stage of the extraction procedure. The function returns the cropped and geometrically normalized face area `Y`.

If you type

```
help register_face_based_on_eyes
```

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

An example of usage of the function is shown below. Here, the function was applied to an sample image that is distributed togethre with the PhD toolbox:

```
eyes.x(1)=160;
eyes.y(1)=184;
eyes.x(2)=193;
eyes.y(2)=177;
X=imread('sample_image.bmp');
Y = register_face_based_on_eyes(X,eyes,[128 100]);
figure,imshow(X,[]);
figure,imshow(uint8(Y),[]);
```

The code sets the coordinates structure `eyes`, reads in the image `X` and based on this input data extracts the face region `Y` from the input image, which it then rescales to a size of 128×100. The result of the presented code is shown in Fig.3.2

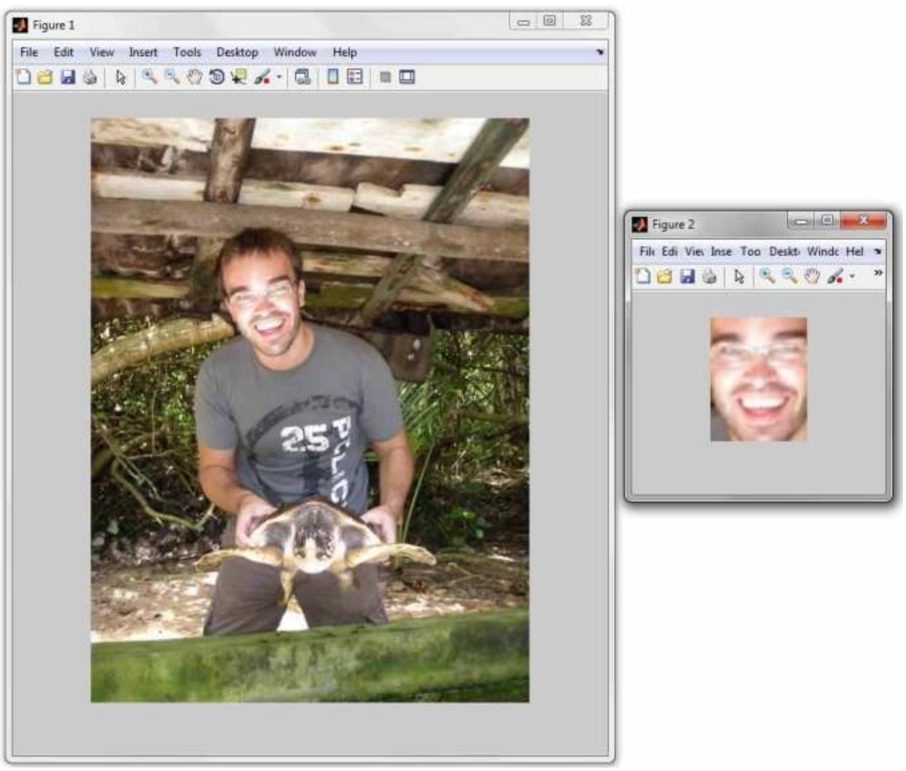


Figure 1
File Edit View Insert Tools Desktop Window Help
Figure 2
File Edi View Inse Too Deskt Windc Hel

Figure 3.2: Results of the sample code

3.1.4 The `resize_pc_comps_PhD` function

The `resize_pc_comps_PhD` function is not intended to be used on its own. It resizes the phase congruency components and concatenates them into a high dimensional feature vector. The description of the function in this document is only present to provide some hints on its functionality. The function has the following prototype:

```
feature_vector = resize_pc_comps_PhD(pc, down_fak);.
```

Here, `pc` denotes the computed phase congruency components that were computed using the function `produce_phase_congruency_PhD`. `down_fak` denotes the down-sampling factor by which to down-sample the phase congruency components before concatenating them into the final `feature_vector`. If you type

```
help resize_pc_comps_PhD
```

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=double(imread('sample_face.bmp'));
filter_bank = construct_Gabor_filters_PhD(8,5, [128 128]);
[pc,EO] = produce_phase_congruency_PhD(X,filter_bank);
feature_vector_extracted_from_X = resize_pc_comps_PhD(pc, 32);
```

The code reads the image named `sample_image.bmp` into the variable `X`, computes phase congruency features from it and employs the `resize_pc_comps_PhD` to construct an feature vector from the computed phase congruency components.

3.2 The *plots* folder

The folder named *plots* contains functions for plotting different performance curves. In the current version of the toolbox three performance curves are supported, namely, ROC, EPC, and CMC curves.The folder contains the following functions:

- `plot_CMC_PhD`,
- `plot_EPC_PhD`, and
- `plot_ROC_PhD`.

A basic description of the functionality of the above listed functions is given in the remainder of this section.

3.2.1 The `plot_CMC_PhD` function

The `plot_CMC_PhD` function plots the so-called CMC (Cumulative Match Score Curve) curve based on the data computed with the `produce_CMC_PhD` function from the PhD face recognition toolbox. The function is very similar to Matlabs' native `plot` function, except for the fact that it automatically assigns labels to the graphs axis and presents the x-axis in a log (base-10) scale. The function has the following prototype:

```
h=plot_CMC_PhD(rec_rates, ranks, color, thickness).
```

If you type

```
help plot_CMC_PhD
```

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:

```
ranks = 1:10;
rec_rates = [0.6 0.7 0.75 0.78 0.8 0.83 0.87 0.95 0.99 1];
h=plot_CMC_PhD(rec_rates, ranks);
axis([1 10 0 1])
```

The code first creates some dummy data for plotting and then plots this data using the `plot_CMC_PhD` function. The results of this sample code is an CMC curve as shown in Fig. 3.3.

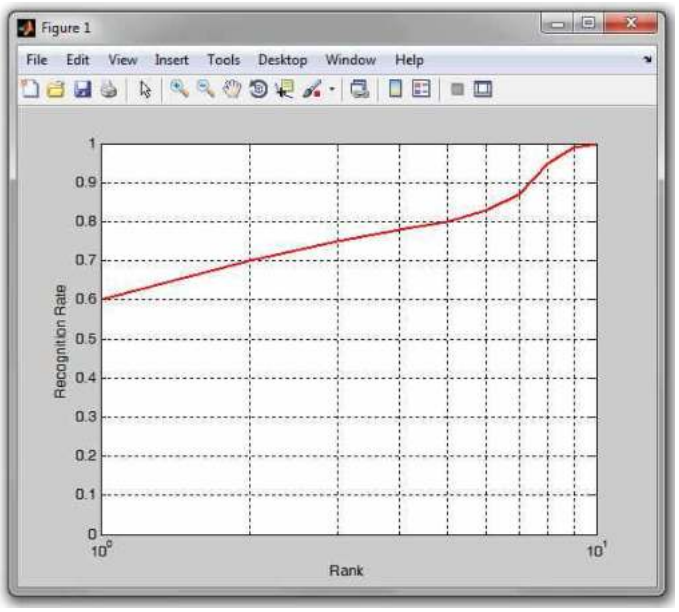


Figure 3.3: An example of a CMC curve created with the sample code.

3.2.2 The `plot_EPC_PhD` function

The `plot_EPC_PhD` function plots the so-called EPC (Expected Performance Curve) curve based on the data computed with the `produce_EPC_PhD` function from the PhD face recognition toolbox. The function is very similar to Matlabs' native `plot` function, except for the fact that it automatically assigns labels to the graphs axis. The function has the following prototype:

```
h=plot_EPC_PhD(alpha,errors, color, thickness).
```

If you type

```
help plot_EPC_PhD
```

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:

```
% we generate the scores through a random generator
true_scores = 0.5 + 0.9*randn(1000,1);
false_scores = 3.5 + 0.9*randn(1000,1);
[ver_rate, miss_rate, rates] = produce_ROC_PhD(true_scores,
false_scores);
true_scores1 = 0.6 + 0.8*randn(1000,1);
false_scores1 = 3.3 + 0.8*randn(1000,1);
[alpha,errors,rates_and_threshs] = produce_EPC_PhD(true_scores,
false_scores,true_scores1,false_scores1,rates,20);
h=plot_EPC_PhD(alpha, errors,'b',4);
```

The code first creates some dummy data for plotting and then plots this data using the `plot_EPC_PhD` function. The results of this sample code is an EPC curve as shown in Fig. 3.4.

3.2.3 The `plot_ROC_PhD` function

The `plot_ROC_PhD` function plots the so-called ROC (Receiver Operating Characteristics) curve based on the data computed with the `produce_ROC_PhD` function from the PhD face recognition toolbox. The function is very similar to Matlabs' native `plot` function, except for the fact that it automatically assigns labels to the graphs axis and presents the x-axis in a log (base-10) scale. The function has the following prototype:

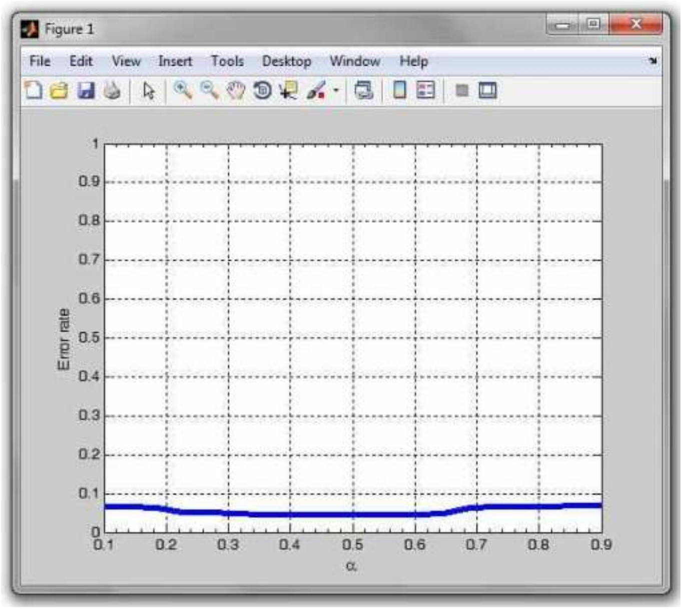


Figure 3.4: An example of an EPC curve created with the sample code.

```
h=plot_ROC_PhD(ver_rate, miss_rate, color, thickness).
```

If you type

```
help plot_ROC_PhD
```

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:

```
true_scores = 0.5 + 0.9*randn(1000,1);
false_scores = 3.5 + 0.9*randn(1000,1);
[ver_rate, miss_rate, rates] = produce_ROC_PhD(true_scores,false_scores);
h=plot_ROC_PhD(ver_rate, miss_rate);
```

The code first creates some dummy data for plotting and then plots this data using the `plot_ROC_PhD` function. The results of this sample code is an ROC curve as shown in Fig. 3.5.

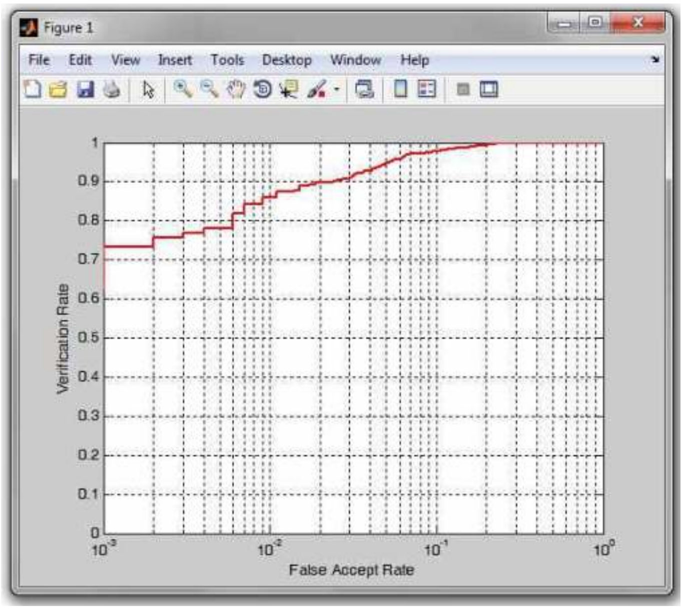


Figure 3.5: An example of a ROC curve created with the sample code.

3.3 The *eval* folder

The folder named *eval* contains functions for constructing different performance curves and evaluating the classification results (creating performance metrics). The folder contains the following functions:

- `produce_EPC_PhD`,
- `produce_CMC_PhD`,
- `produce_ROC_PhD`, and
- `evaluate_results_PhD`.

A basic description of the functionality of the above listed functions is given in the remainder of this section.

3.3.1 The `produce_EPC_PhD` function

The `produce_EPC_PhD` function generates the so-called EPC (Expected Performance Curve) curve from the input data. The generated data can then be used with the `plot_EPC_PhD` function to produce a plot of the EPC curve. The function has the following prototype:

```
[alpha,errors,rates_and_threshs1] = produce_EPC_PhD(true_d,false_d,
true_e,false_e,rates_and_threshs,points).
```

If you type

```
help produce_EPC_PhD
```

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

Note that it not the goal of this document to describe the idea of EPC curves. For that, the interested reader is referred to [2].

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

```
true_scores = 0.5 + 0.9*randn(1000,1);
false_scores = 3.5 + 0.9*randn(1000,1);
[ver_rate, miss_rate, rates] = produce_ROC_PhD(true_scores,false_scores);
true_scores1 = 0.6 + 0.8*randn(1000,1);
false_scores1 = 3.3 + 0.8*randn(1000,1);
[alpha,errors,rates_and_threshs] =
produce_EPC_PhD(true_scores,false_scores,true_scores1,false_scores1,rates,20);
```

The code first creates some dummy data and based on it computes ROC curve parameters. This step can be omitted, but is used to compute the `rates` structure, which is an optional input argument to the `produce_EPC_PhD` and is needed, when some performance metrics (stored in `rates_and_threshs`) should be computed together with the EPC data. In the final stage, the code computes the EPC curve data `alpha` and `errors` as well as some performance metrics (`rates_and_threshs`).

3.3.2 The `produce_CMC_PhD` function

The `produce_CMC_PhD` function generates the so-called CMC (Cumulative Match score Curve) curve from the input data. The generated data can then be used with the `plot_CMC_PhD` function to produce a plot of the CMC curve. The function has the following prototype:

```
[rec_rates, ranks] = produce_CMC_PhD(results).
```

Note here that the output `rec_rates` contains the recognition rates for different ranks. If you are interested in the rank one recognition rate, you can retrieve it by typing:

```
rec_rates(1).
```

If you type

```
help produce_CMC_PhD
```

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:

```
training_data = randn(100,30);
training_ids(1:15) = 1:15;
training_ids(16:30) = 1:15;
test_data = randn(100,15);
test_ids = 1:15;
results = nn_classification_PhD(training_data, training_ids,
test_data, test_ids, size(test_data,1)-1, 'euc');
[rec_rates, ranks] = produce_CMC_PhD(results);
h=plot_CMC_PhD(rec_rates, ranks);
```

The code first creates some dummy feature vectors with corresponding labels, classifies them using the `nn_classification_PhD` function from the PhD toolbox, then produces the CMC curve and finally plots the results.

3.3.3 The `produce_ROC_PhD` function

The `produce_ROC_PhD` function generates the so-called ROC (Receiver Operating Characteristics) curve from the input data. The generated data can then be used with the `plot_ROC_PhD` function to produce a plot of the ROC curve. The function has the following prototype:

```
[ver_rate, miss_rate, rates_and_threshs] = produce_ROC_PhD(true_scores,
false_scores, resolution).
```

Note here that the `produce_ROC_PhD` function in addition to the `rates_and_threshs`, which contains several performance metrics, produces two a vector of verification rates `ver_rate` and a vector of miss rates `miss_rate`, which are used by the `plot_ROC_PhD` to generate a graphical representation of the ROC curve. The plotting function produces a ROC curve that plots the verification rate against the miss rate, with the latter being represented on log (base-10) scale. However, several researcher prefer to plot false acceptance rate (FAR) against the false rejection rate (FRR) using linear scales on both axes. You can do that, by typing:

```
plot(1-ver_rate, miss_rate)
grid on
xlabel('False rejection rate (FRR)')
ylabel('False acceptance rate (FAR)')
```

If you type

```
help produce_ROC_PhD
```

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:

```
true_scores = 0.5 + 0.9*randn(1000,1);
false_scores = 3.5 + 0.9*randn(1000,1);
[ver_rate, miss_rate, rates] = produce_ROC_PhD(true_scores,false_scores,5000);
```

The code first creates some dummy data and then uses the `produce_ROC_PhD` function to compute the ROC curve data.

3.3.4 The `evaluate_results_PhD` function

The `evaluate_results_PhD` functions is one of the more complex functions in the PhD face recognition toolbox. The function provides a way of automatically evaluate the recognition results of your experiments that were produced using the `nn_classification_PhD` function from the toolbox and are stored in an appropriate format. The function itself contains an extensive help section and several examples of usage. Its use is also demonstrated in several demo scripts contained in the *demos* folder of the PhD toolbox.

The function automatically generates all performance curves currently supported in the toolbox and produces several performance metrics for identification as well as verification experiments. All the function does is extracting data from the similarity matrix and meta-data produced by the `nn_classification_PhD` function and calling other evaluation functions from the toolbox. Note that it also produces DET curves if you have installed NISTs DETware before calling the function.

The function has the following prototype:

```
output = evaluate_results_PhD(results,decision_mode,results1).
```

The function produces an output structure `output` that contains the following fields:

```
output.ROC_ver_rate     ...  a vector of verification rates
                             that represents y-axis data of the
                             computed ROC curve
output.ROC_miss_rate    ...  a vector of miss rates (false
                             acceptance rates) that represents
                             x-axis data of the computed ROC
                             curve
output.ROC_char_errors  ...  a structure of characteristic error
                             rates computed from the similarity
                             matrix; it contains the following
                             fields:
```

.minHTER_er	...	the minimal achievable half total error rate
.minHTER_tr	...	the decision threshold that ensures the above minimal half total error rate
.minHTER_frr	...	the false rejection error rate at the minimal Half total error rate
.minHTER_ver	...	the verification rate at the minimal half total error rate
.minHTER_far	...	the false acceptance error rate at the minimal half total error rate
.EER_er	...	the equal error rate
.EER_tr	...	the decision threshold that ensures the above equal error rate
.EER_frr	...	the false rejection error rate at the equal error rate
.EER_ver	...	the verification rate at the equal error rate
.FRR_01FAR_er	...	the half total error rate at the operating point where FRR = 0.1*FAR
.FRR_01FAR_tr	...	the decision threshold that ensures the above half total error rate of FRR = 0.1*FAR
.FRR_01FAR_frr	...	the false rejection error rate at the half total error rate of FRR = 0.1*FAR
.FRR_01FAR_ver	...	the verification rate at the half total error rate of FRR = 0.1*FAR
.FRR_01FAR_far	...	the false acceptance error rate at the half total error rate of FRR = 0.1*FAR
.FRR_10FAR_er	...	the half total error rate at the operating point where FRR = 10*FAR

```
.FRR_10FAR_tr   ...  the decision threshold that ensures
                     the above half total error rate of
                     FRR = 10*FAR
.FRR_10FAR_frr  ...  the false rejection error rate at
                     the half total error rate of FRR =
                     10*FAR
.FRR_10FAR_ver  ...  the verification rate at the half
                     total error rate of FRR = 10*FAR
.FRR_10FAR_far  ...  the false acceptance error rate at
                     the half total error rate of FRR =
                     10*FAR
...
more ...
                      For more information on this
                     structure have a look at the
                     "rates_and_threshs" structure of
                     the "produce_ROC_PhD" function.  The
                     two structures are identical.
```

output.DET_frr_rate	...	a vector of false rejection rates that represent y-axis data of the computed DET curve (this field is only available if NISTs DETware is installed)
output.DET_far_rate	...	a vector of false acceptance rates that represent x-axis data of the computed DET curve (this field is only available if NISTs DETware is installed)
output.CMC_rec_rates	...	a vector of recognition rates that represent y-axis data of the computed CMC curve; NOTE!!: output.CMC_rec_rates(1) represents the rank one recognition rate
output.CMC_ranks	...	a vector of ranks that corresponds to the recognition rates in "output.CMC_rec_rates"; the data represents x-axis data of the computed CMC curve
output.EPC_charq_errors	...	a structure of characteristic error rates on the EPC curve; for a more detailed description of this structure look at the "rates_and_threshs1" output structure of the "produce_EPC_PhD" function; the two structures are identical
output.EPC_alpha	...	a vector of alpha values that represents x-axis data of the computed EPC curve; (this field is only available if two results structures obtained on test as well as evaluation data are provided as input to the function)

```
    output.EPC_errors  ...  a vector of error rates that
                            represents y-axis data of the
                            computed EPC curve; (this field
                            is only available if two results
                            structures obtained on test as well
                            as evaluation data are provided as
                            input to the function)
```

The most important input argument of the function is the `results` structure obtained on some evaluation data using the `nn_classification_PhD` function. The structure contains the some meta-data and more importantly a similarity matrix created during the recognition experiments on the evaluation data. Optionally, the function takes a second `results1` structure obtained on some test data that does not overlap with the evaluation data. This structure is needed when creating EPC curves, but is not compulsory. The last argument of the function is a string called `decision_mode` and takes either the value of `decision_mode='images'` or `decision_mode='ID'`.

In the first case, i.e., `decision_mode='images'`, the performance graphs are cre-

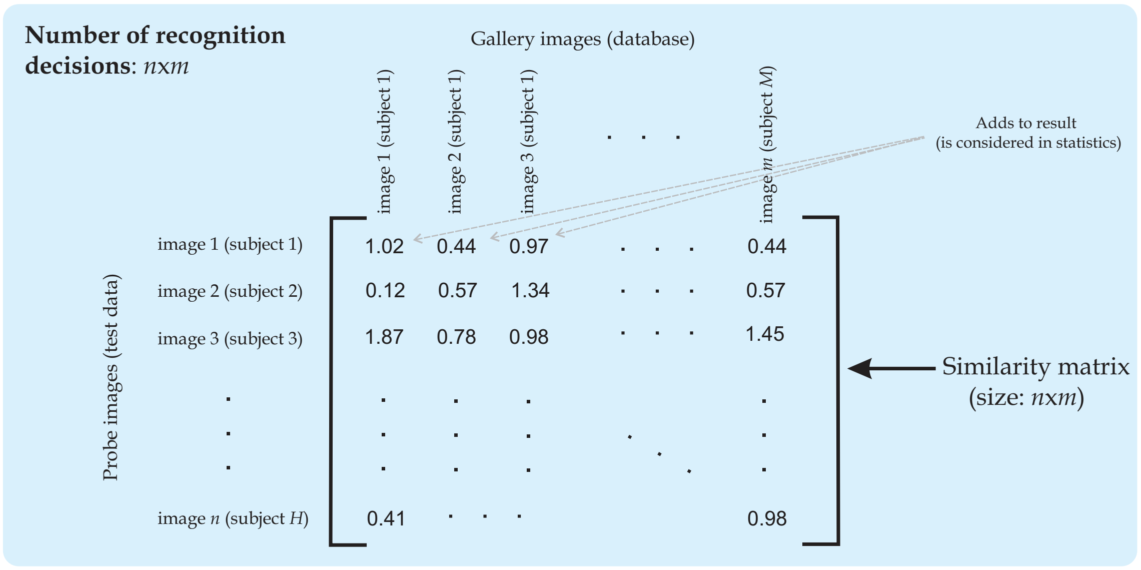


Figure 3.6: Visualization of the effect of the "images" decision mode.

ated as if each image in the database (gallery) is its own template. If we assume that we have n test images (probes) and m gallery images (belonging to M subjects, where $m > M$), then the total number of experiments/recognition attempts in this mode equals nm. Thus, the performance metrics are computed based on pair-wise comparisons. Or in other words, even if we have several images for a given subject available in the database, we conduct recognition experiments for each individual image. The described procedure is visualized in Fig. 3.6.

In the second case , i.e., `decision_mode='ID'`, the performance graphs and metrics are created by treating all images of a given subject as this subjects template. Thus, a given probe image (from some test data) is compared to all images of a given gallery subject and the decision concerning the identity of the probe images is made based on the all comparisons together. If we use the same notation as in the previous example, then the number of experiments/recognition attempts for the "ID" mode equals nM. To put it differently, even if we have several images of one subject in our database, we make only one classification decision. The described procedure is visualized in Fig. 3.7.

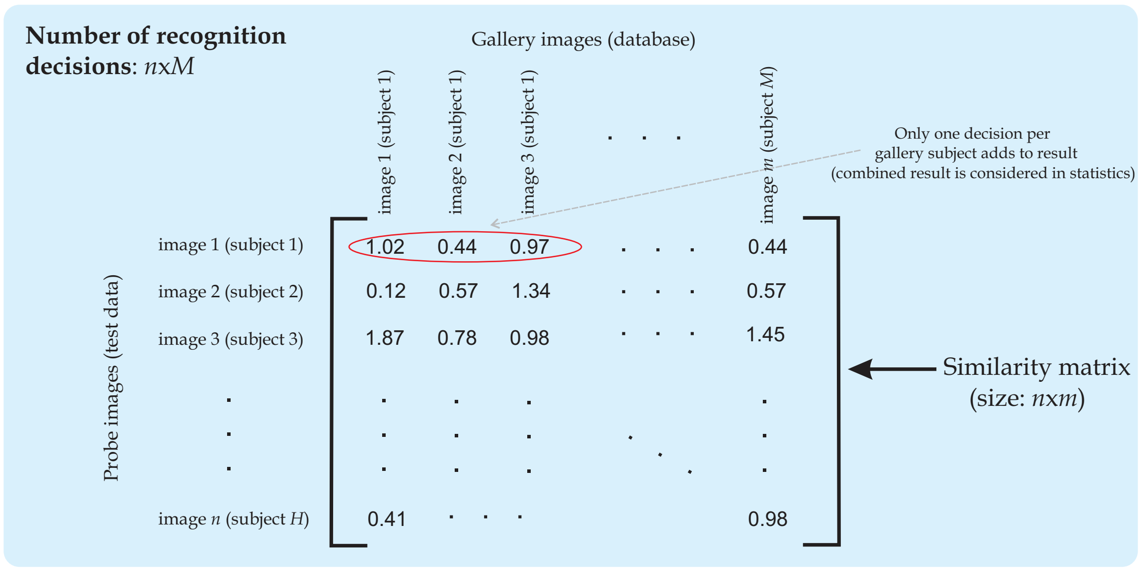


Figure 3.7: Visualization of the effect of the "ID" decision mode.

3.4 The *features* folder

As the name suggests, the folder named *features* contains feature extraction functions. Specifically, the following functions are contained in the folder:

- `filter_image_with_Gabor_bank_PhD`,
- `produce_phase_congruency_PhD`,
- `perform_lda_PhD`,
- `perform_pca_PhD`,
- `linear_subspace_projection_PhD`,

- `perform_kpca_PhD`,
- `perform_kfa_PhD`, and
- `nonlinear_subspace_projection_PhD`.

A basic description of the functionality of the above listed functions is given in the remainder of this section.

3.4.1 The `filter_image_with_Gabor_bank_PhD` function

The `filter_image_with_Gabor_bank_PhD` function filters a facial image with a filter bank of Gabor filters constructed using the `construct_Gabor_filters_PhD` function from the PhD toolbox. The function applies all filters to the input image, computes the magnitude responses, down-samples each of the computed magnitude responses and finally, concatenates the down-sampled magnitude responses into a single feature vector. The function has the following prototype:

```
feature_vector = filter_image_with_Gabor_bank_PhD(image,
filter_bank,down_sampling_factor).
```

Here, `image` denotes the input grey-scale image to be filters, `filter_bank` stands for the Gabor filter bank constructed using the `construct_Gabor_filters_PhD` function, and `down_sampling_factor` stands for the down-sampling factor by which to down-sample the magnitude responses before concatenating them into the resulting feature vector `feature_vector`. Note that this function produces feature vectors such as those used in [11], [6], or [5].

If you type

```
help filter_image_with_Gabor_bank_PhD
```

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:

```
img = imread('sample_face.bmp');
[a,b,c] = size(img);
filter_bank = construct_Gabor_filters_PhD(8, 5, [a b]);
filtered_image = filter_image_with_Gabor_bank_PhD(img,filter_bank,1);
num_of_pixels = sqrt(length(filtered_image)/(8*5));
```

```
figure,imshow(reshape(filtered_image(14*num_of_pixels*num_of_pixels+
1:15*num_of_pixels*num_of_pixels),num_of_pixels,num_of_pixels),[]);
figure,imshow(reshape(filtered_image(37*num_of_pixels*num_of_pixels+
1:38*num_of_pixels*num_of_pixels),num_of_pixels,num_of_pixels),[]);
```

The code reads the image named `sample_face.bmp` into the variable `img`, and construct a filter bank `filter_bank` of 40 Gabor filters and uses the bank to filter the image. During the filtering operation, no down-sampling of the magnitude responses is performed. Thus, we can visualize the magnitude responses at full resolution. This is done in the last two lines of the example code. The result of the above code is a feature vector named `filtered_image`, which can be used as input to subsequent operations (e.g., PCA, LDA, KFA, etc.). The two figures produced by the code are also shown in Fig. 3.8

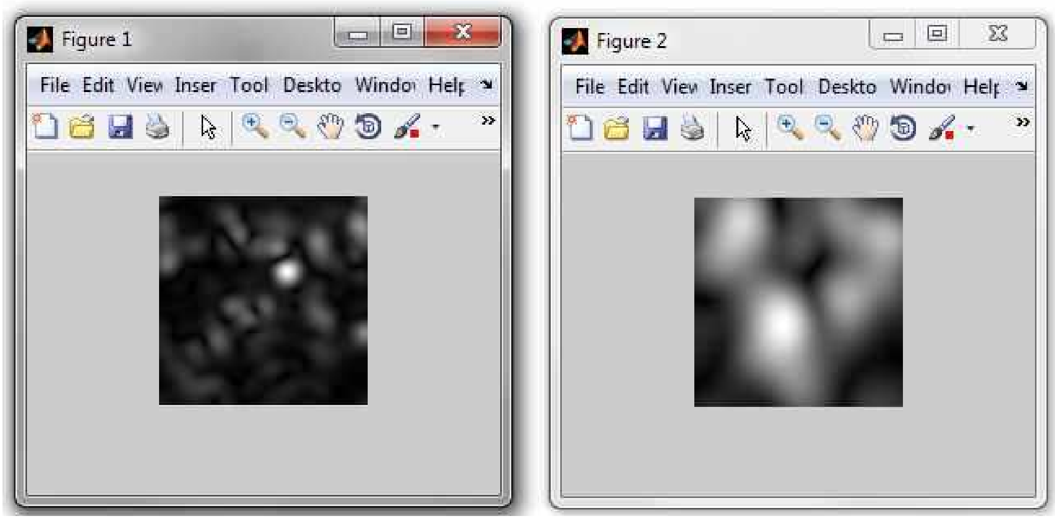


Figure 1
File Edit View Inser Tool Deskto Windo Help
Figure 2
File Edit View Inser Tool Deskto Windo Help

Figure 3.8: Examples of the magnitude responses produced by the example code.

3.4.2 The `produce_phase_congruency_PhD` function

The `produce_phase_congruency_PhD` function is a function that originates from the web page of dr. Peter Kovesi [4]. It was modified to support the filter bank format as used in the PhD toolbox. The function has the following prototype:

`[pc,EO] = produce_phase_congruency_PhD(X,filter_bank,nscale);`.

Here, `X` denotes the input grey-scale image to be filtered, `filter_bank` stands for a Gabor filter bank compatible with the input image `X` (in terms of dimensions) produced with the `construct_Gabor_filters_PhD` function from the PhD toolbox. `nscale` is an optional input argument, which allows to compute the phase congruency features over less scales that are present in the provided filter bank. The function return two output argument, i.e., `pc` and `EO`. The first is a cell array containing the computed phase congruency components for each orientation, while the second is a cell array of complex filter responses.

If you type

```
help produce_phase_congruency_PhD
```

you will get additional information on the function together with several examples of usage. The reader interested in phase congruency and its characteristics is referred to [11] and [3].

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

```
X=double(imread('sample_face.bmp'));
filter_bank = construct_Gabor_filters_PhD(8,5, [128 128]);
[pc,EO] = produce_phase_congruency_PhD(X,filter_bank);
figure(1)
title('Phase congruency at first filter orientation')
imshow(pc{1},[])
figure(2)
title('One of the magnitude responses of the filtering operation')
imshow(abs(EO{5,5}),[])
figure(3)
title('Original input image')
imshow(X,[])
```

The code reads the image named `sample_face.bmp` into the variable `X`, then constructs a compatible filter bank named `filter_bank` and uses this filter bank to compute the phase congruency components. In the last step it shows the original input image, an example of the magnitude response of the filtering and an example of a phase congruency component at a specific orientation. The generated figures are also shown in Fig. 3.9.

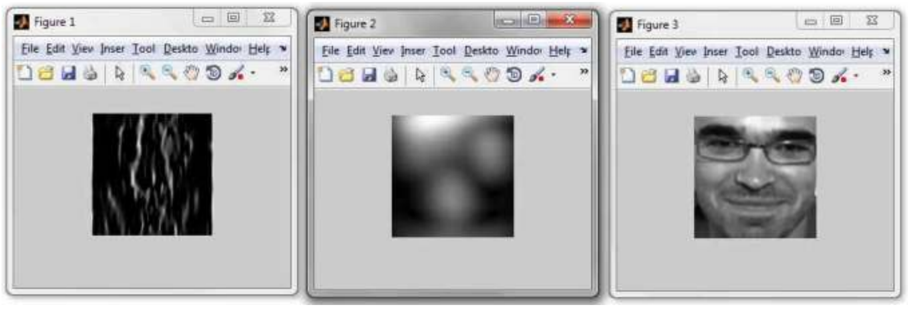


Figure 3.9: The result of the example code: a phase congruency component (left), a magnitude response (middle), the input image (right)

3.4.3 The `perform_lda_PhD` function

The `perform_lda_PhD` function computes the LDA (Linear Discriminant Analysis) subspace from the training data given in the input matrix `X`. The function is basically an implementation of the Fisherface approach, which means that a PCA (Principal component Analysis) step is performed prior to LDA to avoid singularity issues due to a small number of training samples. The function has the following prototype:

```
model = perform_lda_PhD(X,ids,n).
```

Here, `X` denotes an input matrix of training data, `ids` stands for a label vector corresponding to the training data in `X`, and `n` denotes the desired dimensionality of the LDA subspace. Note, however, that `n` cannot exceed $M - 1$, where M is the number of distinct classes (i.e., subject labels) in `X`. The function returns an output structure `model` with the computed subspace and some meta data. For more information on the theoretical background of the Fisherface technique the reader is referred to [1].

If you type

```
help perform_lda_PhD
```

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:

```
%In this example we generate the training data randomly and also
%generate the class labels manually; for a detailed example have a
%look at the LDA demo
```

```
X = randn(300,100); %generate 100 samples with a dimension of 300
ids = repmat(1:25,1,4); %generate 25-class id vector
model = perform_lda_PhD(X,ids,24); %generate LDA model
```

The code first generates some random training data and stores in into a variable called `X`. Next, its generates a corresponding label vector `ids` and finally, it computes the LDA subspace and meta data, which is returned in the output variable `model`.

3.4.4 The `perform_pca_PhD` function

The `perform_pca_PhD` function computes the PCA (Principal Component Analysis) subspace from the training data given in the input matrix `X`. The function is basically an implementation of the Eigenface approach, which was presented in [9]. The function has the following prototype:

```
model = perform_pca_PhD(X,n).
```

Here, `X` denotes an input matrix of training data, and `n` denotes the desired dimensionality of the PCA subspace. Note, however, that `n` cannot exceed $\min d, m-1$, where d is the number of pixels (elements) of a single data sample in the training matrix `X`, and m is the number of images (samples) in the training data matrix `X`.

If you type

```
help perform_pca_PhD
```

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:

```
%In this example we generate the training data randomly and
%set the number of retained components using the actual rank;
%for a detailed example have a look at the PCA demo
X=randn(300,100); %generate 100 samples with a dimension of 300
model = perform_pca_PhD(X,rank(X)-1); %generate PCA model
```

The code first generates some random training data and stores in into a variable called `X`. Next, its generates the PCA subspace and meta data (taking into account the actual rank of the data matrix `X`), which is returned in the output variable `model`.

3.4.5 The `linear_subspace_projection_PhD` function

The `linear_subspace_projection_PhD` function computes the subspace representation of a single test image (sample) or matrix of test images (samples). Hence, it uses the model generated by either the `perform_pca_PhD` function or the `perform_lda_PhD` function to project new test data into the computed subspace, where recognition is ultimately performed. The function has the following prototype:

```
feat = linear_subspace_projection_PhD(X, model, orthogon);.
```

Here, `X` denotes the input test data that needs to be projected into the computed subspace defined by `model`. The third parameter `orthogon` is optional and defines whether the computed subspace in `model` is orthogonal or not. `orthogon` affects the way the subspace projection coefficients are computed and consequently affects the speed of the computation.

If you type

```
help linear_subspace_projection_PhD
```

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:

```
%we randomly initialize the model with all of its fields; in
%practice this would be done using one of the appropriate
%functions
X = double(imread('sample_face.bmp'));
model.P = randn(length(X(:)),1); %random 10 dimensional model
model.dim = 10;
model.W = randn(length(X(:)),10);
feat = linear_subspace_projection_PhD(X(:), model, 0);
```

The code reads the image named `sample_face.bmp` into the variable `X`. Next, its generates a random subspace model and uses this model to compute the subspace representation `feat` of the test image in `X`.

3.4.6 The `perform_kpca_PhD` function

The `perform_kpca_PhD` function computes the KPCA (Kernel Principal Component Analysis) subspace from the training data given in the input matrix `X`. The function is a non-linear variant of PCA and was first presented in [7]. The function has the following prototype:

```
model = perform_kpca_PhD(X, kernel_type, kernel_args,n).
```

Here, `X` denotes an input matrix of training data, `kernel_type` stands for the type of kernel function to use (have a look at the `compute_kernel_matrix_PhD` function), `kernel_args` denotes the arguments of the kernel function, and `n` denotes the desired dimensionality of the KPCA subspace.

If you type

```
help perform_kpca_PhD
```

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:

```
%In this example we generate the training data randomly; for a
%detailed example have a look at the KPCA demo
X=randn(300,100); %generate 100 samples with a dimension of 300
model = perform_kpca_PhD(X, 'poly', [0 2], 90); %generate model
```

The code first generates some random training data and stores in into a variable called `X`. Next, it computes the KPCA subspace and some meta-data, which is returned in the output variable `model`. For constructing the subspace, the code employs a polynomial kernel function.

3.4.7 The `perform_kfa_PhD` function

The `perform_kfa_PhD` function computes the KFA (Kernel Fisher Analysis) subspace from the training data given in the input matrix `X`. The function is a non-linear variant of LDA and was first presented in [5]. The function has the following prototype:

```
model = perform_kfa_PhD(X, ids, kernel_type, kernel_args,n);.
```

Here, `X` denotes an input matrix of training data, `ids` denotes a vector of labels corresponding to the data in `X`, `kernel_type` stands for the type of kernel function to use (have a look at the `compute_kernel_matrix_PhD` function), `kernel_args` denotes the arguments of the kernel function, and `n` denotes the desired dimensionality of the KFA subspace.

If you type

```
help perform_kfa_PhD
```

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:

```
%In this example we generate the training data randomly; for a
%In this example we generate the training data randomly and also
%generate the class labels manually; for a detailed example have a
%look at the KFA demo
X=randn(300,100); %generate 100 samples with a dimension of 300
ids = repmat(1:25,1,4); %generate 25-class id vector
model = perform_kfa_PhD(X, ids, 'poly', [0 2], 24); %generate model
```

The code first generates some random training data `X` and corresponding labels `ids`. It then computes the KFA subspace and returns it together with some meta data in the output variable `model`. For constructing the subspace, the code employs a polynomial kernel function.

3.4.8 The `nonlinear_subspace_projection_PhD` function

The `nonlinear_subspace_projection_PhD` function computes the subspace representation of a single test image (sample) or matrix of test images (samples). Hence, it uses the model generated by either the `perform_kpca_PhD` function or the `perform_kfa_PhD` function to project new test data into the computed subspace, where recognition is ultimately performed. The function has the following prototype:

```
feat = nonlinear_subspace_projection_PhD(X, model);.
```

Here, `X` denotes the input test data that needs to be projected into the computed subspace defined by `model`. The function returns the subspace representation of `X` in the output matrix (or in case of a single input sample in `X` in the output vector) `feat`.

If you type

```
help nonlinear_subspace_projection_PhD
```

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:

```
%we generate random training data for the example
X=double(imread('sample_face.bmp'));
training_data = randn(length(X(:)),10);
model = perform_kpca_PhD(training_data, 'poly', [0 2]);
feat = nonlinear_subspace_projection_PhD(X(:), model);
```

The code reads the image named `sample_face.bmp` into the variable `X` and then generates some training data of the appropriate dimensions. Based on the generated training data, it computes a KPCA subspace and employs the constructed KPCA model to compute the subspace projection coefficients of the input image `X`. The coefficients are returned in the output variable `feat`.

3.5 The *classification* folder

The *classification* folder contains only two functions in the current version of the PhD face recognition toolbox. Specifically, the following two functions are contained in the folder:

- `return_distance_PhD`, and
- `nn_classification_PhD`.

A basic description of the functionality of the above listed functions is given in the remainder of this section.

3.5.1 The `return_distance_PhD` function

The `return_distance_PhD` function computes the distance (or similarity) between two feature vectors. It is possible to use the function on its own if you fill the need for it. The function currently supports four distance/similarity measures, namely, the Euclidian distance, the City block distance, the cosine similarity measure and the cosine mahalanobis (or whitened cosine) similarity measure. The function has the following prototype:

```
d = return_distance_PhD(x,y,dist,covar).
```

Here, `x` and `y` denote feature vectors the similarity of which needs to be computed. `dist` stands for a string ('euc','ctb','cos','mahcos') that specifies, which distance/similarity measure to use and `covar` stands for the covariance matrix of the training data that needs to be provided in case the 'mahcos' similarity measure is selected.

If you type

```
help return_distance_PhD
```

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:

```
%we generate the feature vectors ourselves
x = randn(1,20);
y = randn(1,20);
d = return_distance_PhD(x,y,'euc');
```

The code first generates two random feature vectors `x` and `y` and then calculates the distance `d` between them using the Euclidian distance measure.

3.5.2 The `nn_classification_PhD` function

The `nn_classification_PhD` function performs matching score calculation based on the nearest neighbor classifier. The function has the following prototype:

```
results = nn_classification_PhD(train, train_ids, test, test_ids,
```

`n, dist, match_kind);`.

Here `train` and `test` denote the training (gallery) and testing feature vector matrices, `train_ids` and `test_ids` stand for corresponding label vectors `n` represents the number of feature used in the calculations, `dist` (`'cos'(default) | 'euc' | 'ctb' | 'mahcos'`) denotes the matching distance to use and `match_kind` (`'all' (default) | 'sep'`) represents a string identifier controlling the matching procedure. The function returns a structure `result`, which contains a similarity matrix of the matching procedure and some meta-data in its fields.

If you type

```
help nn_classification_PhD
```

you will get additional information on the function together with several examples of usage. The function features an extensive help section, where all parameters and outputs are described in detail.

Before we proceed to the description of the next function, let us first explore the meaning of the `match_kind` input argument. As already indicated above, the argument can take to values either `'all'`, which is the default or `'sep'`. The first option should be used, when a similarity matrix needs to be constructed, where all feature vectors from the training-feature-vector-matrix `train` are matched against all feature vectors in the test-feature-vector-matrix `test`. The similarity matrix generated by this option is stored in the `results` structure under `results.match_dist`. The structure can then be fed to the `evaluate_results_PhD`, where performance graphs and metrics are computed from the entire similarity matrix. The similarity matrix generates by the `'all'` is visualized in Fig. 3.10.

The second option `'sep'` should be used if you require non-overlapping image sets for clients and impostor in your face verification experiments. In this case, the code generate two similarity matrices as shown in Fig. 3.11.

The first similarity matrix is stored in the `results.client_dist` and the second in the `results.imp_dist` fields of the output `results` structure. These two similarity matrices are used by the `evaluate_results_PhD` function. Here it should be noted, that CMC curves and all performance metrics with respect to identification experiments are generated based on the client similarity matrix. ROC and DET curves and corresponding verification error rates on the other hand are computed by pooling FRR data from the client similarity matrix and

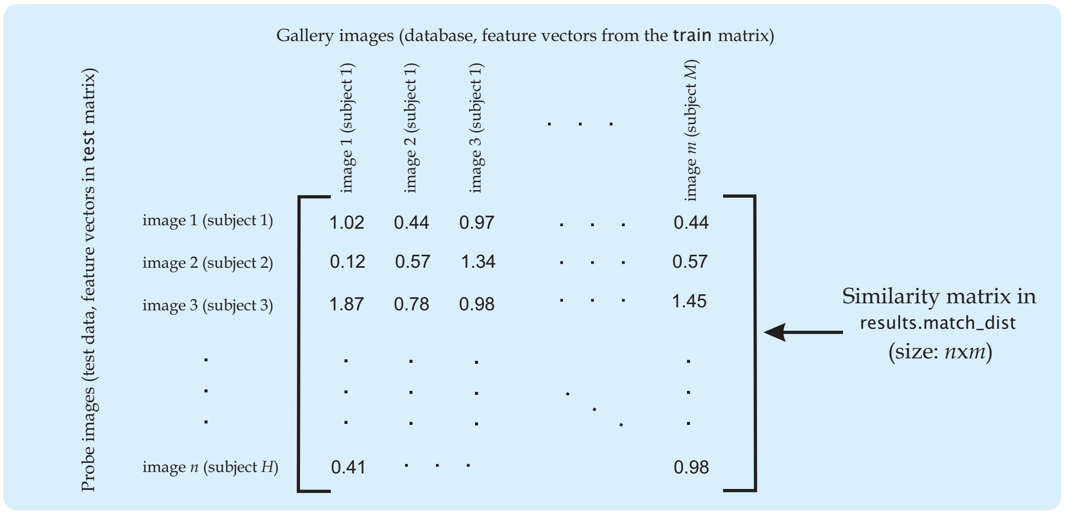


Figure 3.10: An example of the similarity matrix generated by the `all` option.

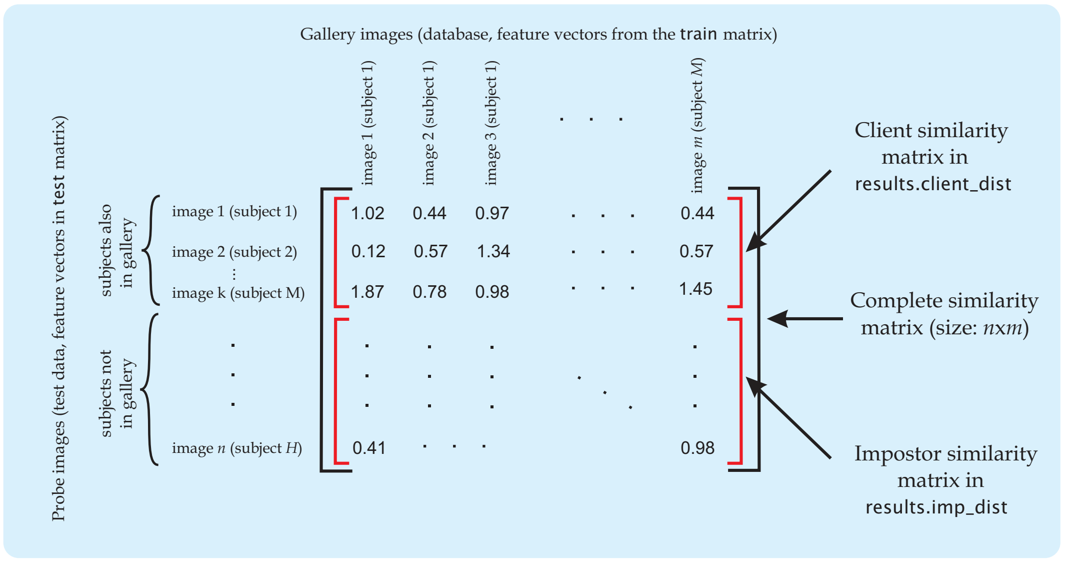


Figure 3.11: An example of the similarity matrix generated by the `sep` option.

FAR data from the impostor similarity matrix.

3.6 The *demos* folder

The *demos* folder is to many probably the most interesting folder in the PhD face recognition toolbox. The folder contains several demo scripts, which demonstrate how to build and evaluate complete face recognition systems and how to test your algorithms on face image databases. In fact, most of the scripts contained in the folder require a database to function. The database is not distributed with the toolbox due to copyright issues, but can be easily downloaded from the web.

In this section we present a detailed description of the individual demo scripts contained in the PhD toolbox (or at least the more interesting ones). These demo scripts comprise:

- `face_registration_demo`
- `construct_Gabors_and_display_bank_demo`
- `Gabor_filter_face_image_demo`
- `phase_cogruency_from_image_demo`
- `perform_pca_demo`
- `perform_lda_demo`
- `perform_kpca_demo`
- `perform_kfa_demo`
- `gabor_PCA_face_recognition_and_evaluation_demo`
- `gabor_LDA_face_recognition_and_evaluation_demo`
- `gabor_KPCA_face_recognition_and_evaluation_demo`
- `gabor_KFA_face_recognition_and_evaluation_demo`
- `PCA_face_recognition_and_evaluation_demo`
- `LDA_face_recognition_and_evaluation_demo`
- `KPCA_face_recognition_and_evaluation_demo`
- `KFA_face_recognition_and_evaluation_demo`
- `phase_congruency_PCA_face_recognition_and_evaluation_demo`
- `phase_congruency_LDA_face_recognition_and_evaluation_demo`
- `phase_congruency_KPCA_face_recognition_and_evaluation_demo`
- `phase_congruency_KFA_face_recognition_and_evaluation_demo`, and
- `preprocessINFace_gabor_KFA_face_recognition_and_evaluation_demo`.

3.6.1 Running the demo scripts (important!!)

It is very important to note that most of the demo scripts will not run of the box. This means that you will have to download and unpack a database of face images, on which the processing can be performed. The demo scripts of the PhD toolbox were written for use with the ORL (AT&T) database, which can be obtained from `http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html`.

You have to download the archive containing the database and unpack it into the `database` folder, which is waiting for you in the *demos* folder of the PhD face recognition toolbox. Hence, after you have unpacked the database, the *database* folder should something like this:

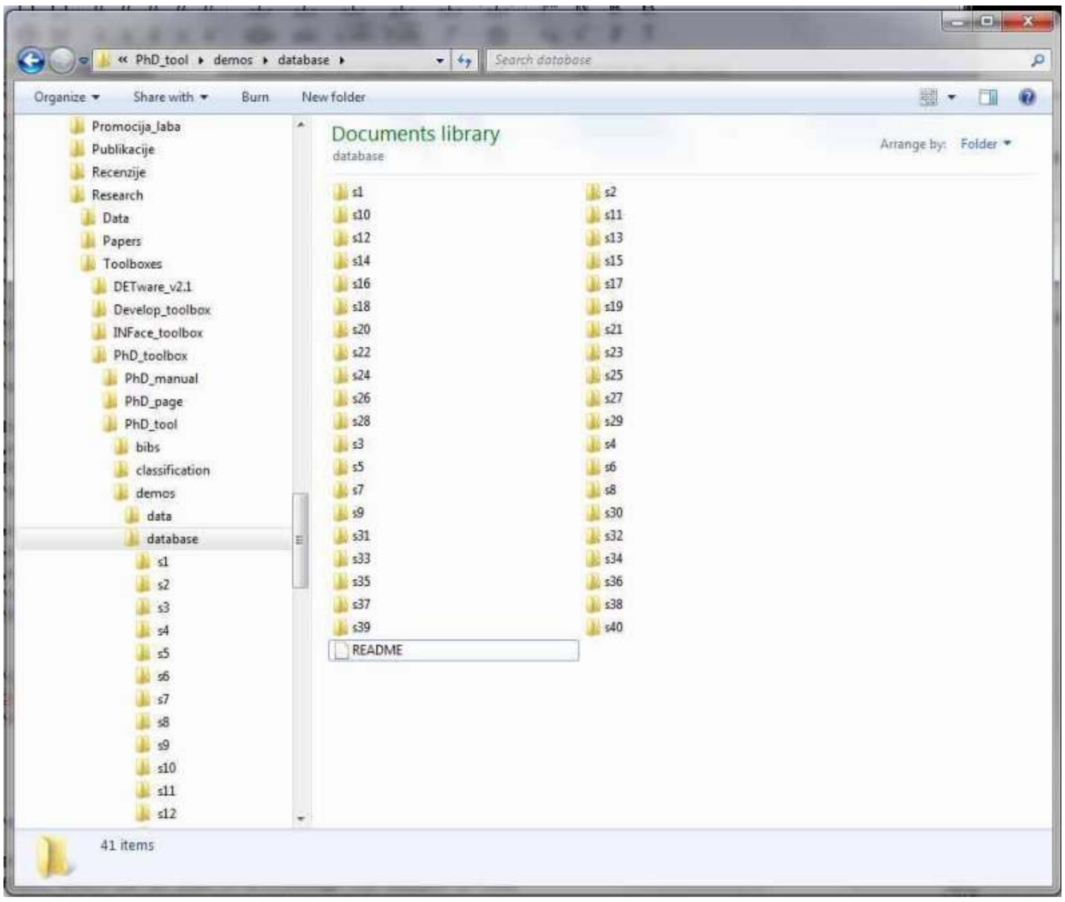


Figure 3.12: The *database* folder should contain 40 subfolders of 40 subjects from the ORL database.

Once you have downloaded and unpacked the database change Matlabs current working directory to the *demos* folder of the PhD face recognition toolbox and run a demo script of your choice. This step is necessary, because the *demos* folder is not added to Matlabs search path, when installing the toolbox, which in turn is a consequence of the fact that all paths referring to the face image data are given in relative terms from the *demos* directory and won't work from any other location.

If you have followed the above instructions all of the demo scripts should work just fine and demonstrate how to use the functions contained in the PhD toolbox.

3.6.2 The `face_registration_demo` script

The `face_registration_demo` script demonstrates how to use the `register_face_based_on_eyes` function to register (or to extract) a face from an input image based on provided eye coordinates. It demonstrates two different examples. In the first the coordinates are hard coded in the script and in the second they are read from an external file. For more information on the demo script have a look at its header.

3.6.3 The `construct_Gabors_and_display_bank_demo` script

The script uses the `construct_Gabor_filters_PhD` function from the PhD toolbox to demonstrate how to construct a bank of Gabor filters for facial feature extraction. The script constructs the default filter bank of 40 filters with 8 orientations and 5 scales, with the filter parameters selected for optimal performance with 128x128 pixel face images. The demo constructs the filter bank and then displays the real parts, imaginary parts and magnitudes of the filters in the bank as shown in Fig. 3.13. For more information on the demo script have a look at its header.

3.6.4 The `Gabor_filter_face_image_demo` script

The script demonstrates how to filter an input image of size 128x128 pixels with a filter bank of 40 Gabor filters. The demo constructs the filter bank, the filter the image with all filter from the bank and finally displays the result of the filtering in a Matlab figure. The entire process is repeated two times with different downscaling factors.

The result of the filtering process presented in the demo script is a set of magnitude responses as shown in Fig. 3.14. For more information on the demo script have a look at its header.

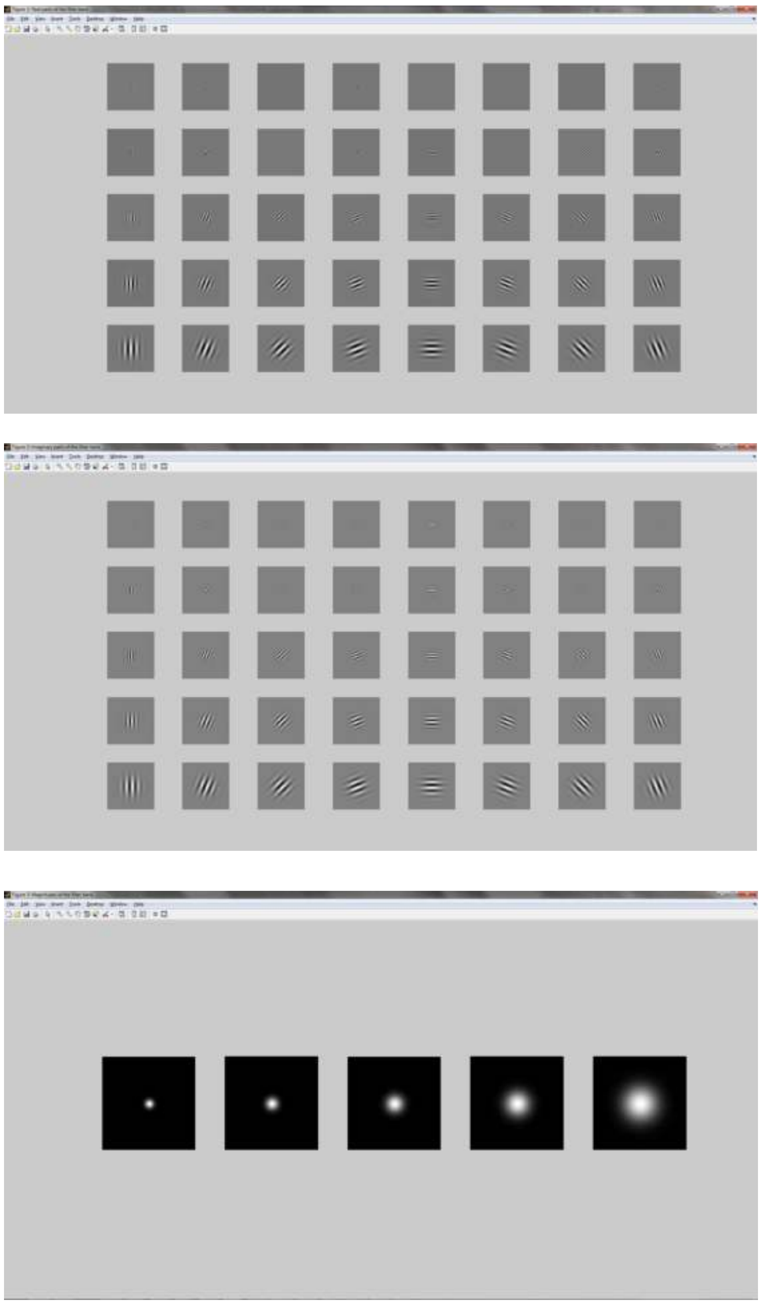


Figure 3.13: Figure generated by the demo script: real parts of the filter bank (top), imaginary parts of the filter bank (middle), magnitudes for different filter scales (bottom)

3.6.5 The `phase_cogruency_from_image_demo` script

The script demonstrates how to compute the phase congruency for an input face image. The demo first construct a filter bank of Gabor filters and then uses this

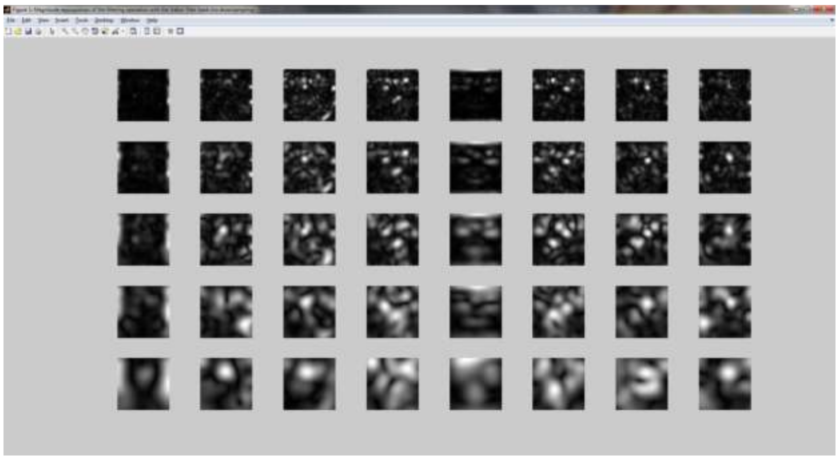


Figure 3.14: Magnitude responses of the filtering process performed by the demo script.

bank to compute the phase congruency for each filter orientation in the filter bank. At the end the results are display in a couple of figures.

The figures generated by the demo script are also shown in Fig. 3.15. For more information on the demo script have a look at its header.

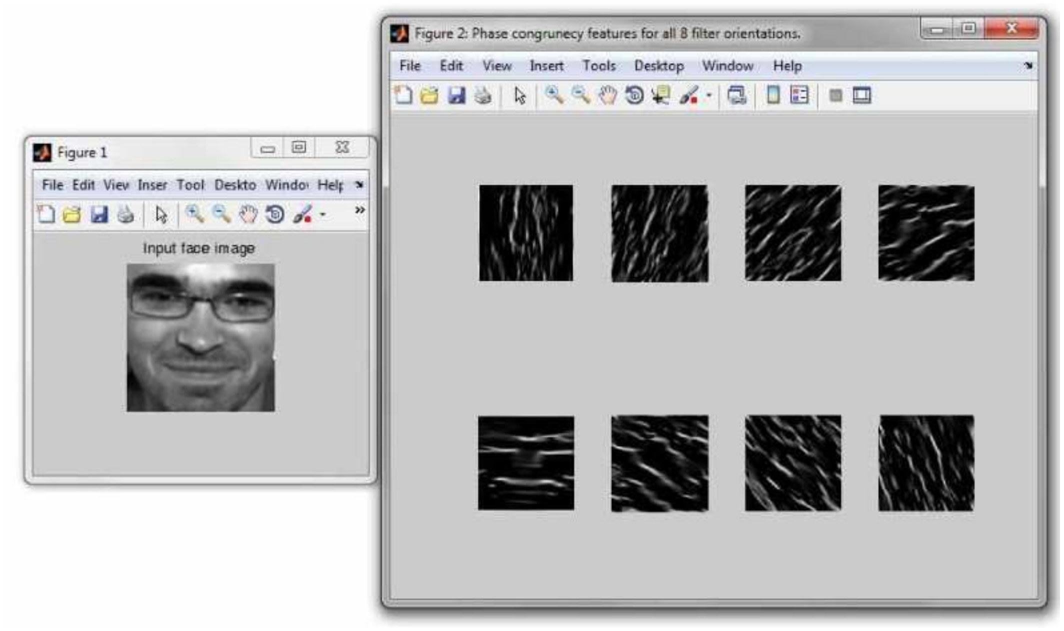


Figure 1
File Edit Viev Inser Tool Deskto Windo Helr
Input face image
Figure 2: Phase congrunecy features for all 8 filter orientations.
File Edit View Insert Tools Desktop Window Help

Figure 3.15: Results of the `phase_cogruency_from_image_demo` script.

3.6.6 The `perform_XXX_demo` scripts

The `perform_XXX_demo` scripts, where XXX stands for either pca, lda, kpca or kfa, demonstrate how to use the subspace projection technique functions[2] from the PhD toolbox. Basically these scripts compute the selected subspace and then show some results.

The demos assume that you have downloaded the ORL database and have unpacked it to the */demos/database* folder. This folder should now have the following internal structure:

```
/demos/database/ — s1/
                 | — s2/
                 | — s3/
                 | — s4/
                 | — s5/
                 | — s6/
                 ...
                 | — s40/
```

Each of the 40 subfolders should contain 10 images in PGM format. If you have not downloaded and unpacked the ORL database at all or have unpacked it into a different folder this demo will not work!!! Please follow the install instructions in the install script or look at Section 3.6.1.

IMPORTANT!!!! Note that you must run all demo scripts in this toolbox from the demos folder. This is particularly important, since some data needed by the scripts is located in folders whose paths are specified relative to the demos folder. If you run the scripts from anywhere else, the scripts may fail.

Since all four functions are relatively similar, we explore the functionality of the `perform_XXX_demo` script family on a specific example and we select the `perform_pca_demo` demo script for this purpose. The script first loads images from the ORL database into a data matrix and then computes the PCA subspace. Ultimately, the projection axes in image form (i.e., the Eigenfaces) and the mean face are displayed in two figures as shown in Fig. 3.16

[2]These functions are: `perform_pca_PhD`, `perform_lda_PhD`, `perform_kpca_PhD`, and `perform_kfa_PhD`

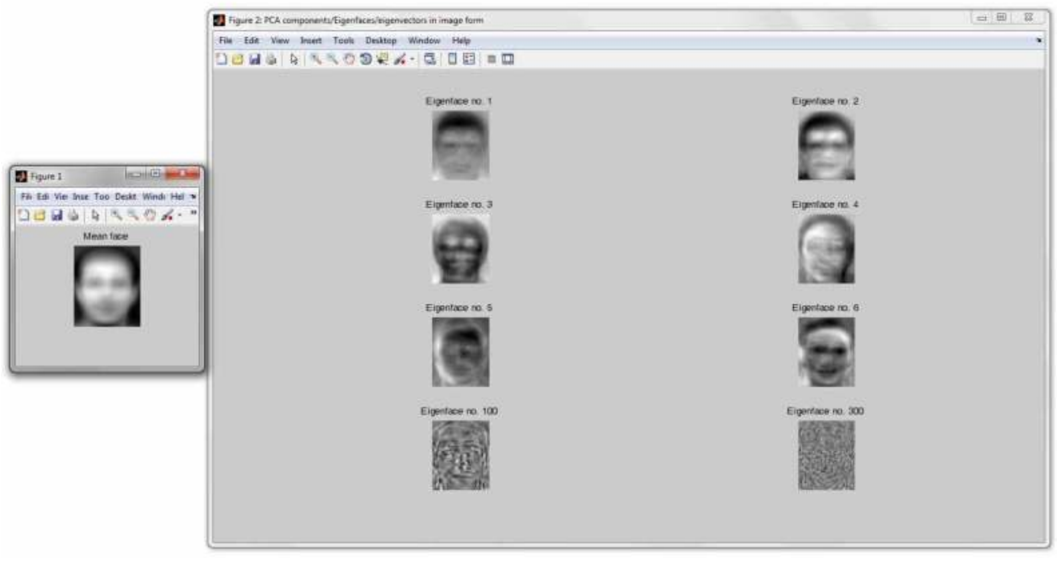


Figure 3.16: Examples of the generated figures: the mean face (left), a few eigenfaces (right).

3.6.7 The `XXX_face_recognition_and_evaluation_demo` scripts

The `XXX_face_recognition_and_evaluation_demo` scripts, where XXX stands for either pca, lda, kpca or kfa, demonstrate how to use the subspace projection technique functions[3] from the PhD toolbox for real face recognition experiments and how to evaluate them on a database. Basically these scripts load facial image data from a database, partition it into appropriate image sets for training and testing, compute the selected subspace, and perform recognition experiments. In the end, the scripts generate several performance curves and performance metrics for the conducted experiments.

The demos assume that you have downloaded the ORL database and have unpacked it to the */demos/database* folder. This folder should now have the following internal structure:

```
/demos/database/ — s1/
                 | — s2/
                 | — s3/
                 | — s4/
                 | — s5/
                 | — s6/
```

[3]These functions are: `perform_pca_PhD`, `perform_lda_PhD`, `perform_kpca_PhD`, and `perform_kfa_PhD`

...
| — *s40/*

Each of the 40 subfolders should contain 10 images in PGM format. If you have not downloaded and unpacked the ORL database at all or have unpacked it into a different folder this demo will not work!!! Please follow the install instructions in the install script or look at Section 3.6.1.

IMPORTANT!!!! Note that you must run all demo scripts in this toolbox from the demos folder. This is particularly important, since some data needed by the scripts is located in folders whose paths are specified relative to the demos folder. If you run the scripts from anywhere else, the scripts may fail.

Since all four functions are relatively similar, we explore the functionality of the `XXX_face_recognition_and_evaluation_demo` script family on a specific example and we select the `LDA_face_recognition_and_evaluation_demo` demo script for this purpose. The script first loads images from the ORL database into a data matrix and then partitions this data into a training and test set. Next, it computes the LDA subspace based on images from the training set and finally performs recognition experiments using images from the test set. In the end, the demo script generates a CMC and ROC curves and displays them in two separate figures as shown in Fig. 3.17. Note that the script also generates a DET curve in case you have installed NISTs DETware.

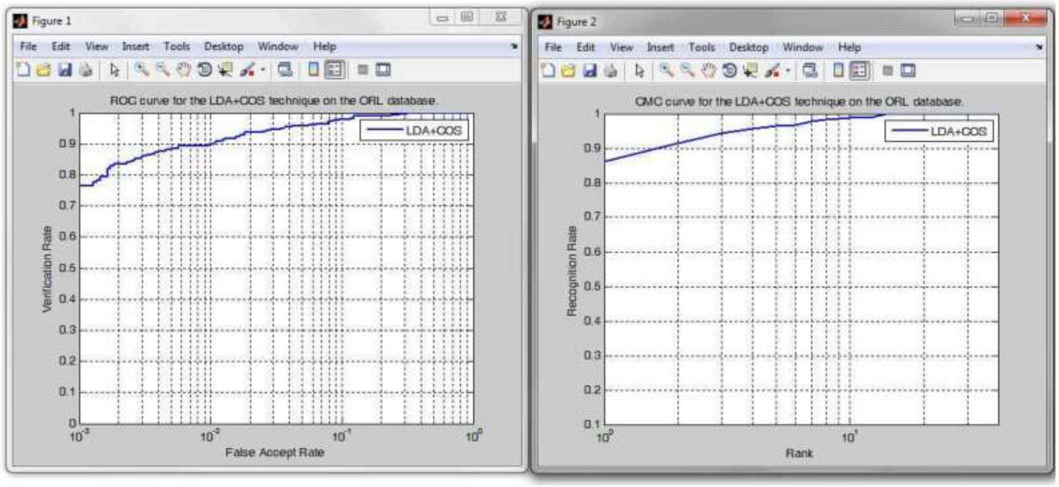


Figure 1
File Edit View Insert Tools Desktop Window Help
ROC curve for the LDA+COS technique on the ORL database.
LDA+COS
Verification Rate
False Accept Rate
Figure 2
CMC curve for the LDA+COS technique on the ORL database.
Recognition Rate
Rank

Figure 3.17: Examples of the generated figures: ROC curve (left), CMC (right).

In addition to the performance curves, the demo scripts also outputs some performance metrics, which should look something like this:

```
===========================================================
SOME PERFORMANCE METRICS:

Identification experiments:
The rank one recognition rate equals (in %):  86.07%

Verification/authentication experiments:
The equal error rate equals (in %):  4.28%
The minimal half total error rate equals (in %):  4.09%
The verification rate at 1% FAR equals (in %):  90.00%
The verification rate at 0.1% FAR equals (in %):  76.43%
The verification rate at 0.01% FAR equals (in %):  64.29%
===========================================================
```

3.6.8 The `gabor_XXX_face_recognition_and_evaluation_demo` scripts

The `gabor_XXX_face_recognition_and_evaluation_demo` scripts, where XXX stands for either pca, lda, kpca or kfa, demonstrate how to use the subspace projection technique functions[4] from the PhD toolbox for real face recognition experiments with Gabor magnitude features. Basically these scripts load facial image data from a database, partition it into appropriate image sets for training and testing, filter the images with a bank of Gabor filters, generate feature vectors of Gabor magnitude features, compute the selected subspace, and perform recognition experiments. In the end, the scripts generate several performance curves and performance metrics for the conducted experiments.

The demos assume that you have downloaded the ORL database and have unpacked it to the */demos/database* folder. This folder should now have the following internal structure:

```
/demos/database/ — s1/
                 | — s2/
                 | — s3/
                 | — s4/
                 | — s5/
```

[4]These functions are: `perform_pca_PhD`, `perform_lda_PhD`, `perform_kpca_PhD`, and `perform_kfa_PhD`

| — *s6/*
...
| — *s40/*

Each of the 40 subfolders should contain 10 images in PGM format. If you have not downloaded and unpacked the ORL database at all or have unpacked it into a different folder this demo will not work!!! Please follow the install instructions in the install script or look at Section 3.6.1.

IMPORTANT!!!! Note that you must run all demo scripts in this toolbox from the demos folder. This is particularly important, since some data needed by the scripts is located in folders whose paths are specified relative to the demos folder. If you run the scripts from anywhere else, the scripts may fail.

Since all four functions are relatively similar, we explore the functionality of the `gabor_XXX_face_recognition_and_evaluation_demo` script family on a specific example and we select the `gabor_KPCA_face_recognition_and_evaluation_demo` demo script for this purpose. The script first loads images from the ORL database into a data matrix and then partitions this data into a training, evaluation and test set. Next, it filters the images using a filter bank of 40 Gabor filters. Based on this process it constructs feature vectors of Gabor magnitude features and computes the KPCA subspace based on feature vectors from the training set and finally performs recognition experiments using feature vectors from the test and evaluation sets. In the end, the demo script generates a CMC, EPC and ROC curves and displays them in three separate figures as shown in Fig. 3.18. Note that the script also generates a DET curve in case you have installed NISTs DETware.

In addition to the performance curves, the demo scripts also outputs some performance metrics, which should look something like this:

```
============================================================
SOME PERFORMANCE METRICS:

Identification experiments:
The rank one recognition rate of the experiments equals (in %):  80.00%

Verification/authentication experiments on the evaluation data:
The equal error rate on the evaluation set equals (in %):  4.17%
The minimal half total error rate on the evaluation set equals (in %):  2.68%
```

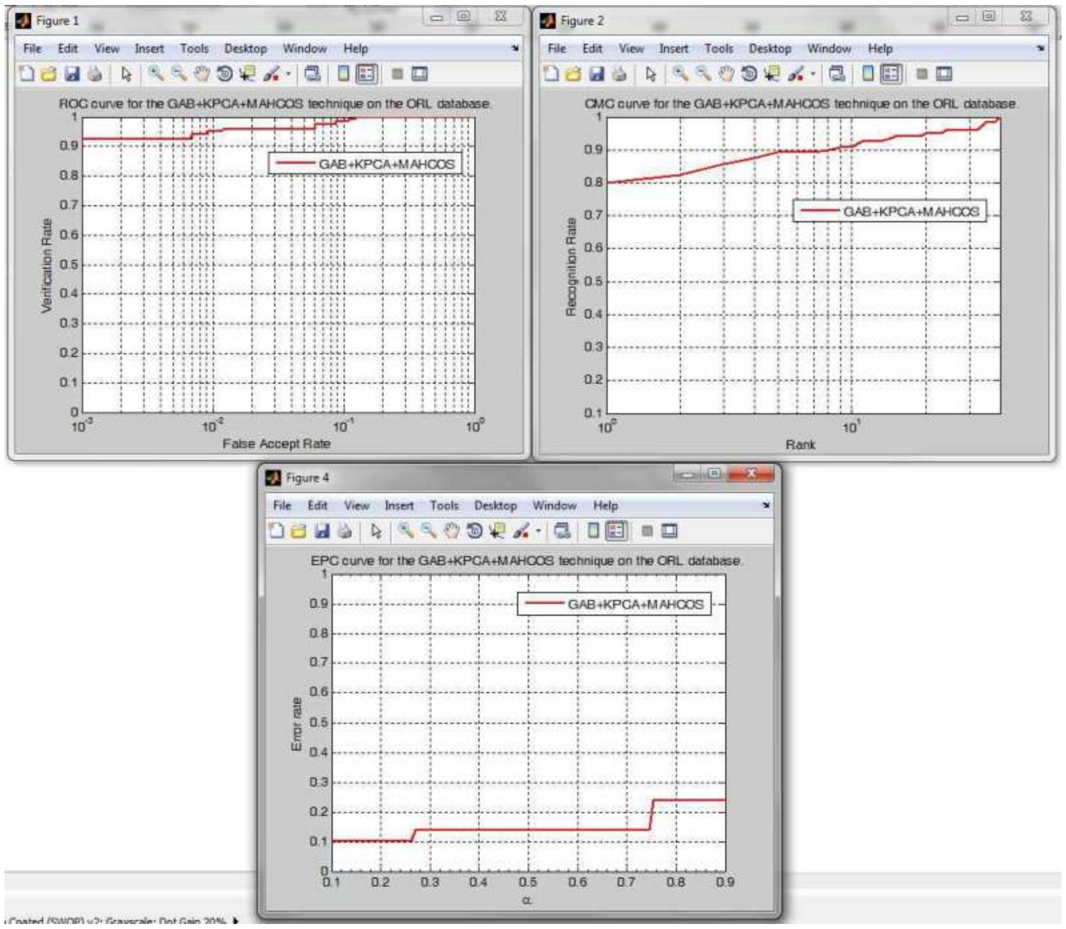


Figure 1
File Edit View Insert Tools Desktop Window Help
ROC curve for the GAB+KPCA+MAHCOS technique on the ORL database.
GAB+KPCA+MAHCOS
Verification Rate
False Accept Rate
Figure 2
CMC curve for the GAB+KPCA+MAHCOS technique on the ORL database.
Recognition Rate
Rank
Figure 4
EPC curve for the GAB+KPCA+MAHCOS technique on the ORL database.
Error rate
α

Figure 3.18: Examples of the generated figures: ROC curve (left), CMC (right), EPC (bottom).

```
The verification rate at 1% FAR on the evaluation set equals (in %): 95.00%
The verification rate at 0.1% FAR on the evaluation set equals (in %): 92.50%
The verification rate at 0.01% FAR on the evaluation set equals (in %): 0.83%

Verification/authentication experiments on the test data (preset thresholds
on evaluation data):
The verification rate at 1% FAR on the test set equals (in %): 71.88%
The verification rate at 0.1% FAR on the test set equals (in %): 56.88%
============================================================
```

The phase_congruency_XXX_face_recognition_and_evaluation_demo scripts

The phase_congruency_XXX_face_recognition_and_evaluation_demo scripts, where XXX stands for either pca, lda, kpca or kfa, demonstrate how to use the subspace projection technique functions[5] from the PhD toolbox for real face recognition experiments with Gabor phase congruency features. Basically these scripts load facial image data from a database, partition it into appropriate image sets for training and testing, filter the images with a bank of Gabor filters, generate feature vectors of Gabor phase congruency features, compute the selected subspace, and perform recognition experiments. In the end, the scripts generate several performance curves and performance metrics for the conducted experiments.

The demos assume that you have downloaded the ORL database and have unpacked it to the */demos/database* folder. This folder should now have the following internal structure:

```
/demos/database/ — s1/
                 | — s2/
                 | — s3/
                 | — s4/
                 | — s5/
                 | — s6/
                 ...
                 | — s40/
```

Each of the 40 subfolders should contain 10 images in PGM format. If you have not downloaded and unpacked the ORL database at all or have unpacked it into a different folder this demo will not work!!! Please follow the install instructions in the install script or look at Section 3.6.1.

IMPORTANT!!!! Note that you must run all demo scripts in this toolbox from the demos folder. This is particularly important, since some data needed by the scripts is located in folders whose paths are specified relative to the demos folder. If you run the scripts from anywhere else, the scripts may fail.

Since all four functions are relatively similar, we explore the functionality of the `phase_congruency_XXX_face_recognition_and_evaluation_demo` script family on a specific example and we select the `phase_congruency_KFA_face_recognition_and_evaluation_demo` demo script for this purpose. The script first loads images from the ORL database into a data matrix and then partitions this data into a training, evaluation and test set. Next, it filters the images using a filter bank of 40 Gabor filters. Based on this process it constructs feature vectors of Gabor phase congruency features and

[5]These functions are: `perform_pca_PhD`, `perform_lda_PhD`, `perform_kpca_PhD`, and `perform_kfa_PhD`

computes the KFA subspace based on feature vectors from the training set and finally performs recognition experiments using feature vectors from the test and evaluation sets. In the end, the demo script generates a CMC, EPC and ROC curves and displays them in three separate figures as shown in Fig. 3.19. Note that the script also generates a DET curve in case you have installed NISTs DETware.

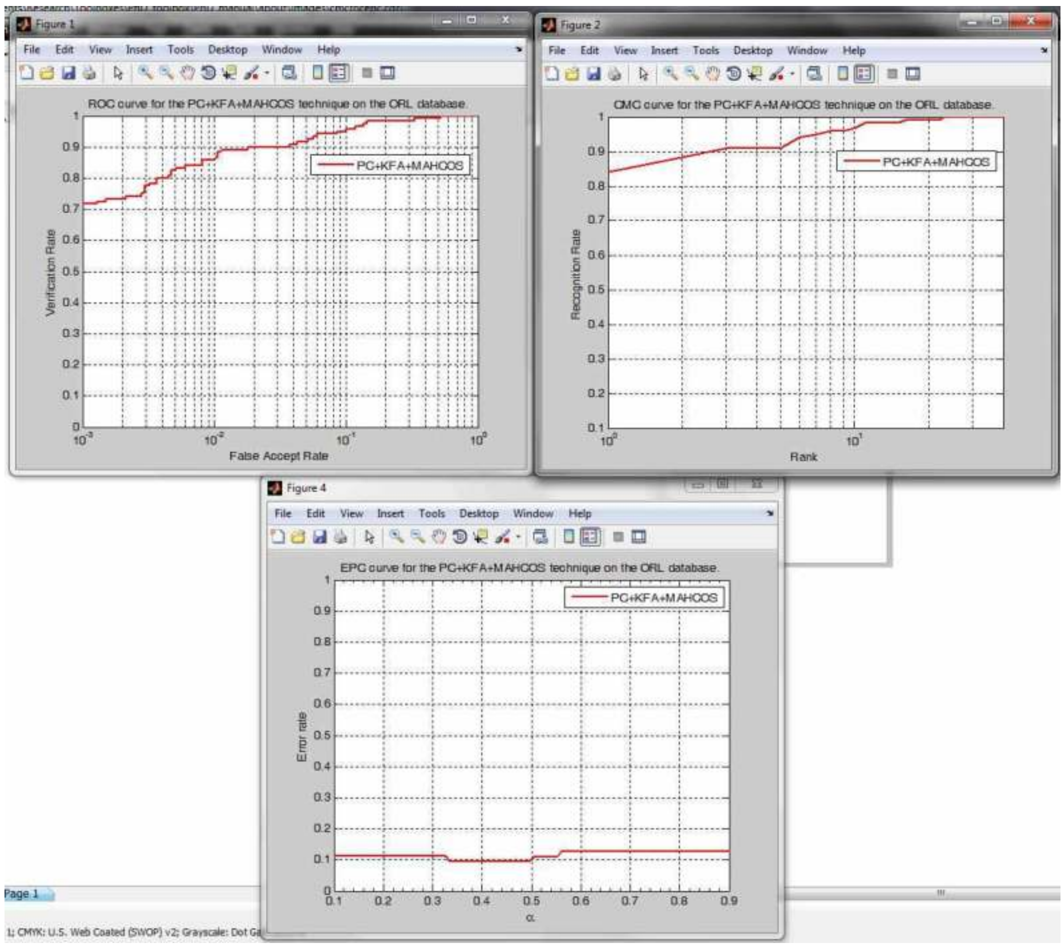


Figure 1
ROC curve for the PC+KFA+MAHCOS technique on the ORL database.
PC+KFA+MAHCOS
Verification Rate
False Accept Rate
Figure 2
CMC curve for the PC+KFA+MAHCOS technique on the ORL database.
Recognition Rate
Rank
Figure 4
EPC curve for the PC+KFA+MAHCOS technique on the ORL database.
Error rate
α

Figure 3.19: Examples of the generated figures: ROC curve (left), CMC (right), EPC (bottom).

In addition to the performance curves, the demo scripts also outputs some performance metrics, which should look something like this:

```
==============================================================
SOME PERFORMANCE METRICS:

Identification experiments:
```

```
The rank one recognition rate of the experiments equals (in %):  84.17%

Verification/authentication experiments on the evaluation data:
The equal error rate on the evaluation set equals (in %):  5.94%
The minimal half total error rate on the evaluation set equals (in %):  5.90%
The verification rate at 1% FAR on the evaluation set equals (in %):  85.83%
The verification rate at 0.1% FAR on the evaluation set equals (in %):  71.67%
The verification rate at 0.01% FAR on the evaluation set equals (in %):  0.83%

Verification/authentication experiments on the test data (preset thresholds
on evaluation data):
The verification rate at 1% FAR on the test set equals (in %):  74.38%
The verification rate at 0.1% FAR on the test set equals (in %):  64.38%
=============================================================
```

The `preprocessINFace_gabor_KFA_face_recognition_and_evaluation_demo` script

The last demo script featured in the PhD face recognition toolbox is the `preprocessINFace_gabor_KFA_face_recognition_and_evaluation_demo` demo scripts. The script demonstrates how to include functions from the INFace toolbox for illumination invariant face recognition in a face recognition system constructed using the functions from the PhD toolbox. Basically these script loads facial image data from a database, partitions it into appropriate image sets for training and testing, preprocesses the images using a photometric normalization technique from the INFace toolbox[6], filters the images with a bank of Gabor filters, generates feature vectors of Gabor magnitude features, computes the KFA subspace, and performs recognition experiments. In the end, the script generates several performance curves and performance metrics for the conducted experiments.

The demo assumes that you have downloaded the ORL database and have unpacked it to the */demos/database* folder. This folder should now have the following internal structure:

/demos/database/ — s1/
| — s2/
| — s3/

[6]You can get the INFace toolbox from [10]

| — *s4/*
| — *s5/*
| — *s6/*
...
| — *s40/*

Each of the 40 subfolders should contain 10 images in PGM format. If you have not downloaded and unpacked the ORL database at all or have unpacked it into a different folder this demo will not work!!! Please follow the install instructions in the install script or look at Section 3.6.1.

IMPORTANT!!!! Note that you must run all demo scripts in this toolbox from the demos folder. This is particularly important, since some data needed by the scripts is located in folders whose paths are specified relative to the demos folder. If you run the scripts from anywhere else, the scripts may fail. You also have to download and install the INFace toolbox or the demo script wont work either.

The script first loads images from the ORL database into a data matrix and then partitions this data into a training, evaluation and test set. Next, it photometrically normalizes the face images using the Tan and Triggs [8] normalization procedure. It then filters the normalized images using a filter bank of 40 Gabor filters. Based on this process it constructs feature vectors of Gabor magnitude features and computes the KFA subspace using feature vectors from the training set. Finally, it performs recognition experiments using feature vectors from the test and evaluation sets. In the end, the demo script generates a CMC, EPC and ROC curves and displays them in three separate figures as shown in Fig. 3.20. Note that the script also generates a DET curve in case you have installed NISTs DETware.

In addition to the performance curves, the demo scripts also outputs some performance metrics, which should look something like this:

```
=============================================================
SOME PERFORMANCE METRICS:

Identification experiments:
The rank one recognition rate of the experiments equals (in %):  95.00%

Verification/authentication experiments on the evaluation data:
```

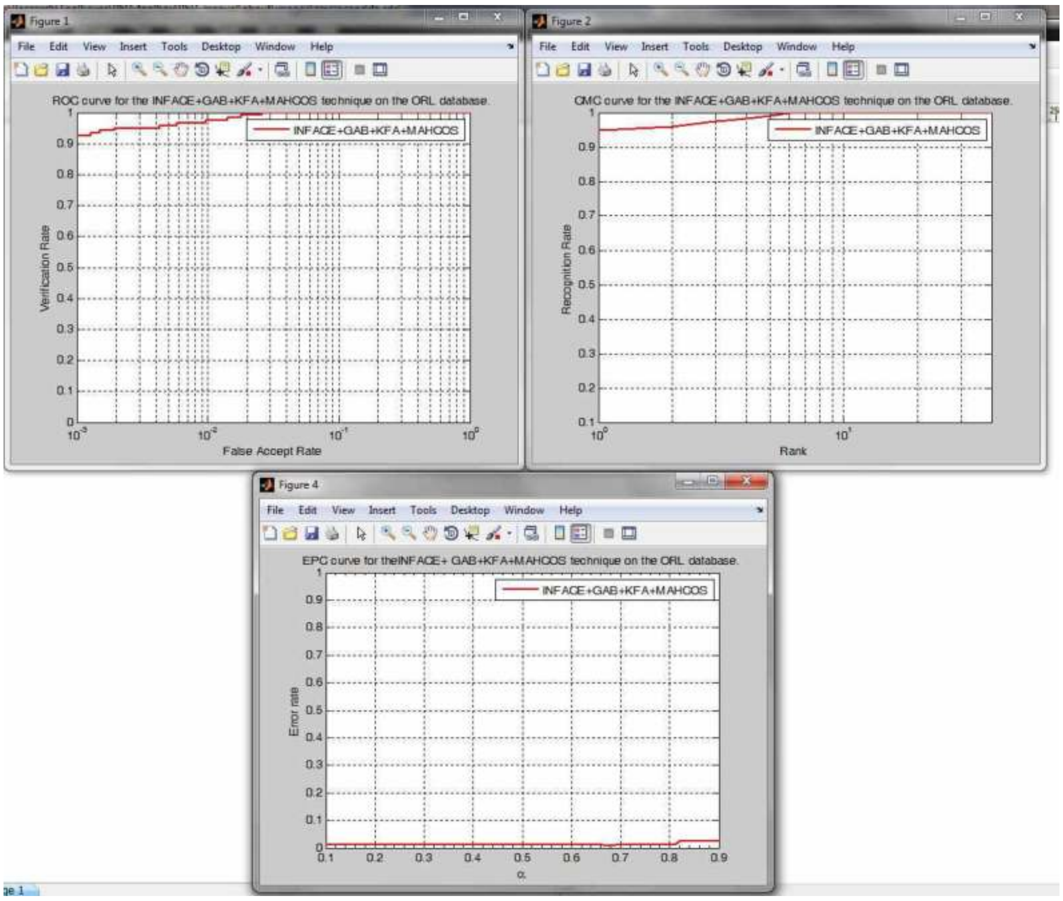


Figure 1
File Edit View Insert Tools Desktop Window Help
ROC curve for the INFACE+GAB+KFA+MAHCOS technique on the ORL database.
INFACE+GAB+KFA+MAHCOS
Verification Rate
False Accept Rate
Figure 2
File Edit View Insert Tools Desktop Window Help
CMC curve for the INFACE+GAB+KFA+MAHCOS technique on the ORL database.
INFACE+GAB+KFA+MAHCOS
Recognition Rate
Rank
Figure 4
File Edit View Insert Tools Desktop Window Help
EPC curve for theINFACE+ GAB+KFA+MAHCOS technique on the ORL database.
INFACE+GAB+KFA+MAHCOS
Error rate
α

Figure 3.20: Examples of the generated figures: ROC curve (left), CMC (right), EPC (bottom).

```
The equal error rate on the evaluation set equals (in %):  1.67%
The minimal half total error rate on the evaluation set equals (in %):  1.30%
The verification rate at 1% FAR on the evaluation set equals (in %):  97.50%
The verification rate at 0.1% FAR on the evaluation set equals (in %):  92.50%
The verification rate at 0.01% FAR on the evaluation set equals (in %):  0.83%

Verification/authentication experiments on the test data (preset thresholds
on evaluation data):
The verification rate at 1% FAR on the test set equals (in %):  98.75%
The verification rate at 0.1% FAR on the test set equals (in %):  93.13%
=============================================================
```

3.7 The ***bibs*** folder

The folder named *bibs* contains only two Latex bib files, which can be used to acknowledge the toolbox in case you are using Latex. The two files are:

- ACKNOWL1.bib, and
- ACKNOWL2.bib.

4. Using the Help

The toolbox contains some basic help which offers additional information on each of the functions and scripts in the PhD face recognition 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 PhD_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 *features* folder:

`help features`.

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`.

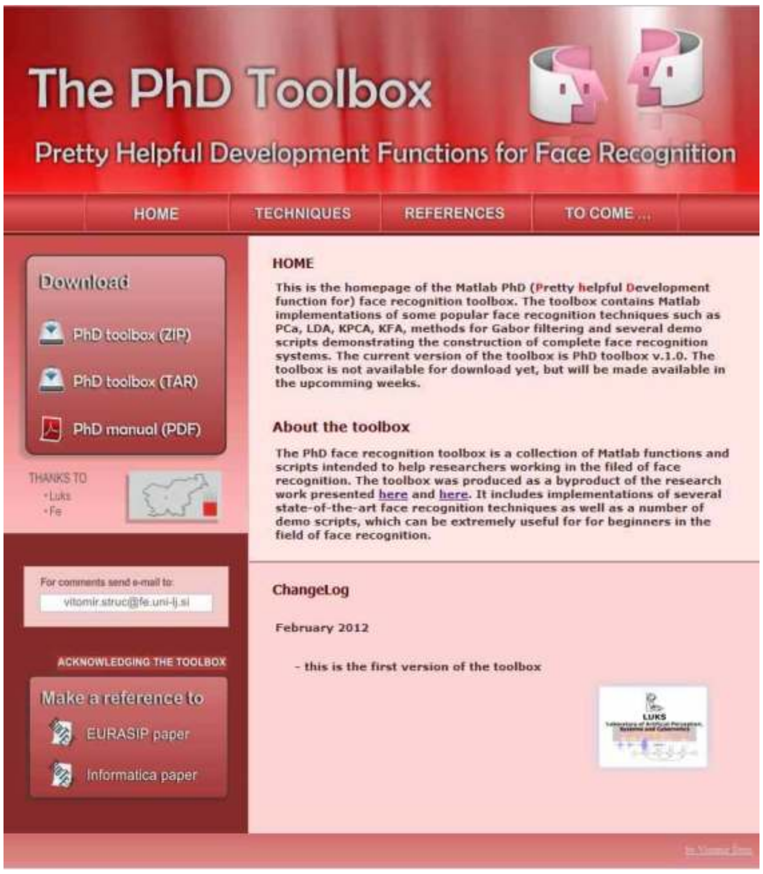
All functions (and scripts) are equipped with an extensive help section describing the functionality of the function and also include a reference to the paper, where the implemented technique was proposed or is explained.

5. The PhD toolbox homepage

In addition to other repositories from where you might have downloaded the PhD face recognition toolbox, the toolbox is also available from its official homepage. The page can be accessed from:

`http://luks.fe.uni-lj.si/sl/osebje/vitomir/face_tools/PhDface/`

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



HOME

This is the homepage of the Matlab PhD (Pretty helpful Development function for) face recognition toolbox. The toolbox contains Matlab implementations of some popular face recognition techniques such as PCa, LDA, KPCA, KFA, methods for Gabor filtering and several demo scripts demonstrating the construction of complete face recognition systems. The current version of the toolbox is PhD toolbox v.1.0. The toolbox is not available for download yet, but will be made available in the upcomming weeks.

About the toolbox

The PhD face recognition toolbox is a collection of Matlab functions and scripts intended to help researchers working in the filed of face recognition. The toolbox was produced as a byproduct of the research work presented here and here. It includes implementations of several state-of-the-art face recognition techniques as well as a number of demo scripts, which can be extremely useful for for beginners in the field of face recognition.

For comments send e-mail to:
vitomir.struc@fe.uni-lj.si

ChangeLog

February 2012

- this is the first version of the toolbox

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

The current version of the PhD toolbox is v1.0, so there so Change Log needed yet.

7. Conclusion

The current version of the toolbox (PhD toolbox v1.0) includes Matlab functions that implement several popular face recognition techniques, evaluation tools, performance graphs, and many more. If you have any questions or useful suggestions with respect to the toolbox you are welcomed to contact me. You can find my contact information by following the following link:

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

Thank you for taking an interest in the PhD face recognition toolbox.

References

[1] P.N. Belhumeur, J.P. Hespanha, D.J. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *Proc. of the 4th European Conference on Computer Vision, ECCV'96*, April.

[2] S. Bengio, J. Mariethoz. The expected performance curve: a new assessment measure for person authentication. *Proceedings of Odyssey 2004: The Speaker and Language Recognition Workshop*, 2004.

[3] P. Kovesi. Image features from phase congruency. *Videre: A Journal of Computer Vision Research. MIT Press*, Vol. 1, No. 3, , 1999. `http://mitpress.mit.edu/e-journals/Videre/001/v13.html`.

[4] P. Kovesi. Computer vision functions for matlab. Website, 2012. `http://www.csse.uwa.edu.au/~pk/Research/MatlabFns/index.html`.

[5] C. Liu. Capitalize on dimensionality increasing techniques for improving face recognition grand challenge performance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 28, No. 5, str. 725–737, 2006.

[6] C. Liu, H. Wechsler. Gabor feature based classification using the enhanced fisher linear discriminant model for face recognition. *IEEE Transactions on Image Processing*, Vol. 11, No. 4, str. 467–476, 2002.

[7] B. Scholkopf, A. Smola, K.R. Muller. Nonlinear component analysis as a kernel eigenvalue problem. Technical Report No. 44, 1996. 18 pages.

[8] 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.

[9] M. Turk, A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neurosicence*, Vol. 3, No. 1, str. 71–86.

[10] V. Štruc. The matlab inface toolbox for illumination invariant face recognition. Website, 2012. `http://luks.fe.uni-lj.si/sl/osebje/vitomir/face_tools/INFace/index.html`.

[11] V. Štruc, N. Pavešić. The complete gabor-fisher classifier for robust face recognition. *EURASIP Advances in Signal Processing*, Vol. 2010, str. 26, 2010.