DeRecLib User's Manual

Companion to the book
OBJECT DETECTION AND RECOGNITION IN DIGITAL IMAGES: THEORY AND PRACTICE
by Boguslaw Cyganek, Wiley 2013

Copyrights 2013
This document is intended only as a companion to the book *Object Detection and Recognition in Digital Images: Theory and Practice* by B. Cyganek (Wiley, 2013) [1]. Its main objective is to provide additional information on software attached to this book (available from the book web page [6]) which was published in an electronic version to keep track of possible changes and updates to the software platform.
# Table of Contents

1 Introduction.......................................................................................................................... 5  
   1.1 What this Technical Report Contains.............................................................................. 5  
   1.2 What is DeRecLib Software .......................................................................................... 5  
   1.3 Software Copyright Note ............................................................................................... 5  

2 Getting Started..................................................................................................................... 6  
   2.1 User Requirements ......................................................................................................... 6  
   2.2 System Requirements .................................................................................................... 6  
   2.3 Development Platform .................................................................................................. 6  
   2.4 Software Architecture ................................................................................................... 7  
   2.5 Installation and the First Build ...................................................................................... 8  
   2.6 Debugging ..................................................................................................................... 9  

3 Software Components ......................................................................................................... 10  
   3.1 Input and Output Operations ........................................................................................ 10  
      3.1.1 Input/Output of the Images ...................................................................................... 10  
      3.1.2 Input/Output of the Video ...................................................................................... 13  
   3.2 Interface to the SVD Computations .............................................................................. 15  
   3.3 Class `EigenImageFor` ................................................................................................... 16  
   3.4 Class `PCA_BackgroundSubtraction` ........................................................................... 19  
   3.5 Examples of Anisotropic Diffusion .............................................................................. 22  
   3.6 Structural Tensor and its Versions .............................................................................. 23  
      3.6.1 Extended and Compact Versions of the Structural Tensor ................................... 23  
   3.7 Testing k-Means Clustering .......................................................................................... 24  
   3.8 Testing Kernels ............................................................................................................. 25  
   3.9 Testing Tensor Decompositions .................................................................................... 25  
   3.10 Working with the HOSVD Based Classifier ................................................................. 28  

4 Summary .............................................................................................................................. 30  

Bibliography........................................................................................................................... 31
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>DMA</td>
<td>Direct Memory Access</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>HIL</td>
<td>Hardware Image Library</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Platform</td>
</tr>
<tr>
<td>OOD</td>
<td>Object-Oriented Design</td>
</tr>
<tr>
<td>OOP</td>
<td>Object-Oriented Programming</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>STL</td>
<td>Standard Template Library</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 What this Technical Report Contains

This document is a supplement to the book *An Introduction to 3D Computer Vision Techniques and Algorithms* by Bogusław Cyganek & J. Paul Siebert (Wiley, 2008) [2]. It contains additional information on inner mechanisms of the library, as well as on practical use of the software platform provided for readers of the book. Therefore it should be read after reading at least Chapter 3 of the aforementioned book which contains information on basic data structures used in the presented library. Because of this, this technical report can be treated as a ‘dynamic’ contents of Chapter 14 of the book [2].

1.2 What is DeRecLib Software

The DeRecLib library contains software components (functions and classes) presented in the book *Object Detection and Recognition in Digital Images: Theory and Practice* [1]. It contains a number of examples to show calling context of the procedures.

DeRecLib does not use OpenCV, although it can be joined to that library through the adapters which translate matrix (image) formats between the two platforms. Details of such connection are described in the HIL manual [5].

1.3 Software Copyright Note

The software [6] accompanying the book *Object Detection and Recognition in Digital Images: Theory and Practice* by Boguslaw Cyganek (Wiley, 2013) [1] is supplied only for educational and/or academic purposes and to accompany the book. All other commercial and/or non-private applications require written permission. The software is supplied as it is without any guarantees or responsibility for its use in any application.

If software or its parts are used in scientific publications, the citation of the book is greatly acknowledged.
2 Getting Started

The goal of this chapter is to provide basic information necessary to install, build, run, and debug the projects included in the software available from the web site of the book [6].

2.1 User Requirements

The only requirement is basic knowledge of the C++ language.

The basics theory of software operation of the methods in the DeRecLib are described in the book Object Detection And Recognition in Digital Images: Theory and Practice [1]. Thus, it is advisable to at least skim over the related sections of this book. If one is interested in definition of basic data structures and base operations, then the book An Introduction to 3D Computer Vision Techniques and Algorithms can be recommended [2].

Since the Visual Studio 2010 is used, a basic knowledge of this platform is also useful.

2.2 System Requirements

In order to be able to run the attached software examples it is necessary to have installed the following software components on the computer:

- Windows 7 or higher.
- Microsoft Visual Studio 2010 or higher.

However, software is written in the plain C++ and therefore it can be also compiles on Linux.

2.3 Development Platform

The main development platform is Visual Studio 2010 by Microsoft®, or higher. However, the source code is written in the style which allows its easy porting to other platforms, such as Linux. Therefore only standard C++ features and libraries are used [10][8].

Visual Studio provides also easy to use editor and debugger.
2.4 Software Architecture

Figure 2.1 presents the components diagram which visualizes relationships between the DeRecLib and other utility components.

The attached software relies on a number of external libraries from which the most important are

- HIL - this is our previous library which supplies all definitions of the images and matrices, as well as the most basic operations (such as convolution, morphology, etc.). This library is described in [2][5].
- EigenLib - used to compute SVD decomposition of matrices [3].
- TIFF - used to open and save TIFF files [9].
- JPEG - used to open and save JPEG files [9].

As already mentioned, the most important dependence of the DeRecLib is the HIL library, described in the previous book [2], as well as in the Internet documentation [5]. HIL provides the basic definitions of the image data structures, as well as basic operations such as convolutions and morphological operations. However, DeRecLib utilizes only a fraction of this functionality.
As shown in Figure 2.1 the DeRecLib library can be also connected with the OpenCV library through the interface in the HIL platform, as described in the manual available at [5].

2.5 Installation and the First Build

Installation consists in downloading the .zip'ped repository and unpacking it into your disk. Make sure that after this all files have the read/write attribute set to 'on'. After unpacking, the following directories should be visible in the chosen master directory:

- DeRecLib
- HIL
- JPEG
- Tiff
- TestData

The main software is contained in the DeRecLib directory. Its structure is as follows:

- Platforms
- Source

The TestData directory contains the test files which are used in the examples placed in the DeRecLib library. The output directory is (where results are written):

   DeRecLib\Platforms\Windows\NET 2010\DeRecLib

To access this directory from the source files, the common prefix string is defined as follows (main.cpp):

```cpp
// This string defines a common path to the test images
const string kTestImageFolder( ".\\..\\..\\..\\..\ TestData\" );
```

If it happens that data files are in other directory, this string should be changed appropriately to point to that directory tree.

The projects require installed Microsoft Visual C++ 2010 or higher. Other IDEs can also be used. These, however, can require some changes to the organization of the project and/or minor modifications to the source files.

---

1 The used images come from the following repositories:

- Face images from Georgia Tech® face database
- Face images from the AT&T Laboratories Cambridge® database (formerly the database from Olivetti Research Lab (ORL)).
After installing, load the project

Platforms\Windows\NET 2010\DeRecLib.sln

then launch "Build" and then "Run" commands. The project should get built and executed. If there are some problems with this, first thing to check are setting of the paths of the header files. These are available after choosing the "Project : DeRecLib Properties ..." menu.

There are two versions of the project:

- Debug
- Release

The first one allows code debugging, however at a cost of run-time operation. It has to be remembered that some tensor decomposition procedures require high amount of memory and quite a lot of time to finish, especially if run in Debug mode. Also in Debug mode the REQUIRE assertion is active.

The Release mode is optimized for speed of execution. It should be built and launched after the code has been well checked in the Debug mode.

Finally, let us notice that if possible the x64 version should be chosen. This is especially important when trying to allocated large buffers of memory for tensor decompositions.

### 2.6 Debugging

Some hints on code debugging are described in the manual attached to the HIL platform (available at [5]). When running DeRecLib example functions it may be the case that some REQUIRE assertions will not hold. Most probably this happens if some accuracy of a tensor decomposition is placed in such an assertion. Such situation is harmless, since it indicates that the desired accuracy of decomposition cannot be met for the chosen parameters. Usually the code can be launched to execute further procedures. Nevertheless, each REQUIRE which does not hold should be checked and the problem fixed.

Debugging greatly facilitates code understanding. Therefore each interested procedure should be in-depth debugged to grasp their main steps.
3 Software Components

In this section discussed are software components which are new, auxiliary, or additional to the ones described in the book.

3.1 Input and Output Operations

Input and output functions are auxiliary in order to be able to show functionality of the procedures presented in the book. However, although they require many files, this is not a primary functionality of the DeRecLib library which internally operates not on JPEG, TIFF or other image formats but on image and vide objects defined in the HIL library. Therefore, if used in other projects, the IO functionality can be removed and substituted with other IO modules.

3.1.1 Input/Output of the Images

Input and output operation are necessary to test operation of basic procedures with real data examples. However, their usage is cumbersome since frequently they are heavily platform dependant. In order to avoid such limitations, special input/output function interfaces were created to accompany the main exemplary procedures. These are summarized below (for details see the .\DeRecLib\Source\PRIMotion\VideoHelpers.h header).

There are two basic functions to load monochrome and color images, respectively. These are defined as follows:

```cpp
//===--------------------------------------------------------------===
// This function
// tries to create a monochrome image from that file
//===--------------------------------------------------------------===
//
// INPUT:
//   fileName - full path name of the file to open
//
// OUTPUT:
//   auto ptr to the monochrome image, or
//   0 if failure
//
// REMARKS:
//
// MIAP OrphanMonochromeImageFromFile( const string & fileName );
```
Both functions try to figure out a type of an image to open. This can be either JPEG or TIFF. Other types are not supported. Their only parameter fileName should contain either the full path to a file, or a path to a file relative to the current directory. Both functions return auto-pointers to the monochrome or color images, respectively. However, a calling function should always check if a returned pointer is different from 0 which indicates function failure.

It is also possible to explicitly load either JPEG or TIFF files. To load and store the images saved in a JPEG format we use the following functions:

```cpp
// This function
// tries to create a monochrome image from that file
////////////////////////////////////////////////////////////////////////
//
// INPUT:  
//   fileName - full path name of the file to open
//
// OUTPUT: 
//   auto ptr to the color image, or 
//   0 if failure
//
// REMARKS: 
//
CIAP OrphanColorImageFrom_JPEG_File( const string & fileName );
```

```cpp
// This function
// tries to create a monochrome image from that file
////////////////////////////////////////////////////////////////////////
//
// INPUT:  
//   fileName - full path name of the file to open
//
// OUTPUT: 
//   auto ptr to the monochrome image, or 
//   0 if failure
//
// REMARKS: 
//
MIAP OrphanMonochromeImageFrom_JPEG_File( const string & fileName );

CIAP OrphanColorImageFrom_JPEG_File( const string & fileName );
```
void Save_JPEG_Image(const MonochromeImage & outImage, const string & defaultName = "MonoImage");

void Save_JPEG_Image(const ColorImage & outImage, const string & defaultName = "ColorImage");

Analogously, to load and store the images saved in the TIFF format the following functions should be used:

MIAP OrphanMonochromeImageFrom_TIFF_File( const string & fileName );

CIAP OrphanColorImageFrom_TIFF_File( const string & fileName );
Examples of using the above functions can be observed in the attached code. These can be easily found with help of the "Find in Files" function (Ctrl+Shift+F).

### 3.1.2 Input/Output of the Video

It is assumed that each video sequence is stored as a series of frames, each also in JPEG or TIFF formats. Their names should end with consecutive numbers. For example, the frames should be named as image_0, image_1, image_2, and so on. The whole series can be loaded into a single `MonochromeVideo` or `ColorVideo` objects (defined in the HIL platform), using one of the following functions.
MonochromeVideo * CreateAndOrphan_MonoVideo_FromFiles(
    const string & commonFileName,
    const string & fileTypeSuffix = "jpg",
    int frameNumStart = 0, int frameNumEnd = 100);

ColorVideo * CreateAndOrphan_ColorVideo_FromFiles(
    const string & commonFileName,
    const string & fileTypeSuffix = "jpg",
    int frameNumStart = 0, int frameNumEnd = 100);

Important assumption here is that the commonFileName parameter should contain the "common" name of all the frames without the ending sequence number and extension. That is, for the above example, we should place "image_". If the sequence is in some other directory, then this should be preceded with the appropriate path name, such as "D:\Research\"
There are also two specialized versions named `DisplayAllFramesFrom` to display all frames in a video and in a tensor. In the latter case, each slice of a tensor is treated as an image.

```cpp
template<typename T>
void DisplayAllFramesFrom(const TFlatTensorFor<T> & inTensor, const char * text2Display = "TensorFrame");
```

```cpp
template<typename PIX>
void DisplayAllFramesFrom(const TVideoFor<PIX> & theVideo, const char * commonName = "Frame");
```

3.2 Interface to the SVD Computations

As shown in the diagram in Figure 2.1, DeRecLib utilizes the Eigen library which details can be found on the Internet site [3]. This is an object-oriented library which allows computation of the SVD decomposition in many variants. Also important is possibility of using user defined data types, such as the TFixedFor defined in HIL for representation of the fixed-point data and arithmetic [5].
However, the Eigen library uses different matrix format than HIL. Therefore, there is the adapter (template) class named Compute_SVD_For. Its details are available in the ../DeRecLib/Source/Auxiliary/Math/SVD/Compute_SVD.h file.

3.3 Class EigenImageFor

PCA decomposition is described in Section 3.3.1 of the book. The PCA_For class (Section 3.3.1.1) computes PCA decomposition of any data. However, it is optimized for rather low-dimensional data types. In other words, this is the $L\leq N$ case (for $L$ - data dimensionality, $N$ - number of data). On the other hand, many methods assume computation of PCA decomposition for data being vectorized versions of the images. In this case we have $L\gg N$ (see pg. 209 of the book). The template class EigenImageFor implements just the latter case.

Figure 3.1. A class hierarchy of the PCA and EigenImageFor.

The most important members of the EigenImageFor class are listed in Algorithm 3-1,
template <typename IN_DATA_TYPE, typename NUMERIC_TYPE>
class EigenImageFor : public PCA_For< IN_DATA_TYPE, NUMERIC_TYPE >
{
    protected:
        Matrix * fX_AV; // in columns of this matrix
        // already zero-mean data points
        // are stored

    public:
        typedef typename PCA_For< IN_DATA_TYPE, NUMERIC_TYPE > BaseClass;

    public:
        EigenImageFor( void ) : fX_AV( 0 ) {}
        virtual ~EigenImageFor() { delete fX_AV; }

        public:

        /////////////////////////////////////////////////////////////////////////////
        // These functions computes the covariance matrix sigma
        /////////////////////////////////////////////////////////////////////////////
        //
        // INPUT:
        //
        //    inVideo - reference to the series of frames (a video)
        //
        //    in which each frame contains ONE data point,
        //    i.e. the whole image
        //
        //    theMean - on success contains the mean data point
        //
        //    orphanCovarMatrix - on success this is computed
        //    covariance matrix (an orphaned object) but
        //    of the fX_AV(T) * fX_AV
        //
        //
        // OUTPUT:
        //
        //    true on success
        //    false otherwise
        //
        // REMARKS:
        //
        //    orphanCovarMatrix should be deleted by the calling
        //    function
        //
        virtual bool Compute_CovarianceMatrix( const Video & inVideo,
                DataPoint & theMean,
                Matrix * & orphanCovarMatrix );
These functions computes the principal component decomposition (called: PCA).

**INPUT:**
- `inVideo` - reference to the series of frames (a video) in which each frame contains one feature of data (i.e. it is only one component of the feature vector).
- `orphanCovarMatrix` (optional) - object of input data points at an orphaned covariance matrix.
- `orphanEigenvalues` (optional) - on success points at orphaned matrix with eigenvalues set in descending order (actually this is a one row image).

**OUTPUT:**
- `true` on success
- `false` otherwise

**REMARKS:**
- It computes and saves the `fMean` member which contains the mean point.
- `orphanEigenvalues, orphanCovarMatrix` should be deleted by the caller.

```cpp
virtual bool Compute_PCA( const Video & inVideo,
                          Matrix * & orphanCovarMatrix,
                          Matrix * & orphanEigenvalues );
```

**Algorithm 3.1.** Definition of the `TFlatTensorFor` class for tensor representation in the matricized (flattened) form (the most important members shown). A tensor is defined providing its series of indices, flattening mode, and index permutation (cyclic) mode.

The outlined class `EigenImageFor` plays a major role in the PCA based background method described in Section 3.3.1.3 of the book. The class for this method is described in the next section.
3.4 Class *PCA_BackgroundSubtraction*

Algorithm 3-2 presents definition of the *PCA_BackgroundSubtraction* class with the most important members shown. As already mentioned, the class allows background subtraction.

```cpp
///==================================
/// This class implements a background
/// detector based on fast PCA
///==================================
class PCA_BackgroundSubtraction {
  public:
    typedef MonochromeImage::PixelType      PixelType;
    typedef TImageFor< PixelType >          ImageType;
    typedef auto_ptr< ImageType >           IMAP;
    typedef TVideoFor< PixelType >          VideoType;

    typedef double ArithmType;
// this type is used for all intermediate computations
    typedef EigenImageFor< PixelType, ArithmType > EigenImageProcessorType;

    typedef MonochromeImage                 BackgroundMapImageType;
    typedef auto_ptr< BackgroundMapImageType > BMAP;

    typedef TVideoFor< ArithmType >         ArithmVideoType;
// for scalar video objects
    typedef auto_ptr< ArithmVideoType >     RVAP;

  protected:
    VideoType *                           fInputVideo;
// this object contains the input video
// it is NOT managed by this class. However it can be get/set if necessary
    EigenImageProcessorType               fEigenImageProcessor;
// this object does efficient PCA

  public:

    enum { kMinNumOfFramesForPCA = 3 };  // at least this number of frames to compute PCA
    enum { kBackgroundVal = 0, kForegroundVal = -1 };  

  private:
    int                                    fBackgroundModel_NumOfFrames;
```
public:

///////////////////////////////////////////////////////////
// Default constructor
///////////////////////////////////////////////////////////

// INPUT:
// inputVideo - a pointer to the orphaned video
// object which specified frames will be used to
// build the background model

// OUTPUT:
// none

// REMARKS:
// The input video will be destroyed by the
// class destructor unless released
// with ReleaseInputVideo

PCA_BackgroundSubtraction( VideoType * inputVideo = 0 )
    : fInputVideo( inputVideo ), fBackgroundModel_NumOfFrames( 0 )
{
}

public:

///////////////////////////////////////////////////////////
// This function builds the background model by constructing
// the PCA from the selected frames of the input video
///////////////////////////////////////////////////////////

// INPUT:
// from_frame_index - start frame from the input
// video to be used in the model (default 0)
// to_frame_index - one-after-the-last frame
// to be used (-1 by default, means the very
// end of the video)

// OUTPUT:
// true if ok
// false otherwise

// REMARKS:
// The input video should be set by calling
// the SetInputVideo() member

virtual bool BuildBackgroundModel( int from_frame_index = 0,
                                   int to_frame_index = -1 );

////////////////////////////////////////////////////////////
// This function projects the input image onto the
// background model to build a background map
////////////////////////////////////////////////////////////

20
Algorithm 3-2. Definition of the PCA_BackgroundSubtraction class with the most important members shown.

Examples of background subtraction are presented in Section 3.3.1.3 of the book (pg. 214). These can be generated with help of the attached code in the DeRecLib with the function BackgroundSubtraction_PCA_Test, declared as follows:
3.5 Examples of Anisotropic Diffusion

Anisotropic diffusion is discussed in Section 2.6.2 of the book. Based on theoretical derivations, the implementation of the diffusion processes was organized in a class hierarchy - see Section 2.6.3 (pg. 40). The main class is the AnisotropicDiffusion.

Anisotropic diffusion can be used in many practical applications, such as for filtering in the extended and compact structural tensors (Section 2.7.4), or to build the nonlinear scale-space used for dental implant recognition (Section 5.4.3).

The DeRecLib contains the following function to test basic behavior of the anisotropic diffusion:

```cpp
void AnisotropicDiffusion_Test( const string & testFileName, const string & resultFileName, const int max_iterations = 100 );
```
It is called from the main function and the results are saved to the current output directory. Since the anisotropic diffusion performs iteratively, the procedure accepts also the maximally allowed number of iterations.

Yet a different version of the AnisotropicDiffusion is attached to the software. It allows signal smoothing, however the smoothing control function is driven by a gradient of other signal. Such version was used to anisotropically smooth a probability field controlled with the intensity signal of the original image. The implementation is included in the AnisotropicDiffusion_WithDifferentControl class.

### 3.6 Structural Tensor and its Versions

Structural tensor is discussed in Section 2.7 of the book. Its properties are also presented in the previous book [2]. Implementation of the structural tensor is contained in the HIL library [5].

#### 3.6.1 Extended and Compact Versions of the Structural Tensor

Implementation of different versions of the structural tensor is presented in Section 2.7.4.1 of the book. DeRecLib contains two functions (.\DeRecLib\Source\Tensor\ST\ExtCompactStructTensor_Test.cpp):

```c++
void TensorAnisotropicExtended_Test( const string & testFileName, int max_iterations = 25 );
```
// This function computes the compact structural tensor from the input image.

double TensorAnisotropicCompact_Test(const string & testFileName, int max_iterations = 25, int numOfImportantComponents = 2 );

The above are launched in the main function to test the extended and compact structural tensor, respectively.

### 3.7 Testing k-Means Clustering

The following function was added to show basic properties of the k-means family of objects.

```cpp
// This function does k-means clustering of the input data file (data of any length).

// INPUT:
// fileName - name of the text file (ASCII) with data to be clustered. Each data is stored in a separate line.
// expected_clusters - the assumed number of clusters

// OUTPUT:
// none
```
It is called from the main function with the data file containing skin samples in the RGB space. Results are saved to the output files (their names are placed in the `Test_k_Means` function and can be easily changed).

The above function creates the basic `k_Means` object. However, in its place any other object from the hierarchy can be used (see Section 3.11 of the book).

### 3.8 Testing Kernels

A simple test for the Gaussian kernel contains the following function (`../DeRecLib/Source/Auxiliary/Kernels/KernelTest.cpp`):

```cpp
void GaussianKernel_TestFunction( void );
```

### 3.9 Testing Tensor Decompositions

There is a number of simple test functions which show basic calling mechanisms to different decompositions of tensors. Some of their parameters were hard coded to simplify the interfaces. These can be easily changed. The first from the mentioned test functions does rank-1 tensor approximation:
The next function carries out the rank-1 decomposition of tensors for different numbers of elements in the series (see Section 2.12.6 of the book):

```cpp
void Best_Rank_1_DECOMPOSITION_TEST( const string & tensor_FileName );
```

Best rank-R is tested in the following function (see Section 2.12.7 of the book):

```cpp
void Best_Rank_R_DECOMPOSITION_TEST( const string & tensor_FileName );
```
void Test_Best_Rank_R( const string & inTensor_FileName );

For simplicity, results of the all above test functions are saved to the text files in the current directory.

The last one does video tensor compression, as discussed in section 2.12.8.2 of the book. It allows reproduction of the images shown in Figure 2.37 of the book. Its declaration looks like the following:

string C_o_l_o_r___Rank_R_Compression_Test( const string & colorVideoFileName );

The requested ranks of the output tensor are set in the function code. However, it has to be remembered that this procedure can require substantial amounts of memory, as well as long time to complete (from few minutes, up to hours, depending on the size of the input tensor, requested ranks, and certainly computational power of the computer).
3.10 Working with the HOSVD Based Classifier

The HOSVD based classifier is described in Section 5.8 of the book. However, the HOSVD decomposition is presented in Section 2.12.2, while the HOSVD induced tensor bases are discussed in Section 2.12.4. In this section we present two functions to train and to run the multi-class HOSVD classifier, respectively.

To perform the HOSVD classification three steps are necessary:

1. Initialization of the \_HOSVD MultiClass\_Classifier object, as well as paths to directories containing training images - each directory for a separate class.
2. Training of the HOSVD multi-classifier with the HOSVD\_Classifier\_TRAIN\_Test function.
3. Classification of unknown pattern with the HOSVD\_Classifier\_RUN\_Test function.

Their declarations are as follows (see ..\DeRecLib\Source\PR\HOSVD Classifier\HOSVD\_Classifier\_Test.cpp).

```cpp
// This function trains the HOSVD classification with multiple classes. For each class a separate HOSVD space is built (see Section 5.8 of the book).
bool HOSVD\_Classifier\_TRAIN\_Test(
    T\_HOSVD\_MultiClass\_Classifier & HOSVD\_MultiClassifier,
    vector< string > & video_paths );
```

The training function accepts the already created T\_HOSVD\_MultiClass\_Classifier object as well as a vector with paths to the directories containing patterns (images). Each directory contains exemplars of a single class. Numbers of patterns for each class can be different, however.

```cpp
// This function does HOSVD classification among multiple classes. For each class a separate HOSVD space
```
On the other hand, the HOSVD_Classifier_RUN_Test function should receive the already trained T_HOSVD_MultiClass_Classifier object as its first argument and an unknown test pattern (image) as its second argument, respectively.

In the attached example the ORL ATT face library was used. First five directories were used to train the HOSVD classifier. In each directory there is 10 images, from which 9 is taken for training and the remaining one for testing. In this simple example HOSVD attains 100% accuracy.


4 Summary


DeRecLib contains all code fragments which were presented in the aforementioned book. Additionally it contains a number of test functions which show application of these procedures in straightforward calling scenarios. The main idea was to present basic functionality with relatively small amount of code, thus facilitating its understanding and possible modifications. Therefore the examples run in a console mode. However, they can be easily moved to other applications and platforms, such as Linux.

The platform is planned to be extended. For all further questions, hints on possible improvements, bug reports, etc. please contact myself at: cyganek (at) agh.edu.pl
### Bibliography


[5] [www.wiley.com/go/cyganek3dcomputer](http://www.wiley.com/go/cyganek3dcomputer)

[6] [www.wiley.com/go/cyganekobject](http://www.wiley.com/go/cyganekobject)


Kraków, 2013