## Contents

1 Facial Expression Tracking Application (FETA) Dokumentation .................................................. 1
   1.1 Introduction ......................................................................................... 1
   1.2 Requirements ................................................................................... 1
   1.3 Installation ....................................................................................... 2
   1.4 Walkthrough .................................................................................... 3

2 Facial Expression Tracking Application (FETA) Data Structure Documentation .............................. 5
   2.1 AAM Class Reference ........................................................................ 5
   2.2 AAM_Facade Class Reference .......................................................... 8
   2.3 AAM_POIC Class Reference .............................................................. 19
   2.4 AAM_SIC Class Reference ................................................................. 21
   2.5 AffineMesh Class Reference ............................................................ 23
   2.6 ANN_Facade Class Reference .......................................................... 25
   2.7 ASM Class Reference ................................................................. 29
   2.8 ASM_Facade Class Reference .......................................................... 36
   2.9 FaceDetect Class Reference ............................................................ 45
   2.10 GUIManager Class Reference ......................................................... 47
   2.11 GUIMenue Class Reference ............................................................ 54
   2.12 IFacade Class Reference ................................................................. 57
   2.13 LinearModel Class Reference ......................................................... 59
   2.14 PDM Class Reference ..................................................................... 60
   2.15 Shared Class Reference ................................................................. 74
   2.16 Stopwatch Class Reference ............................................................ 77
Chapter 1

Facial Expression Tracking Application (FETA) Dokumentation

1.1 Introduction

FETA is an application to classify facial expressions.

It presents how the combination of statistical methods, here the Active Shape Model [1] and Active Appearance Model [2], and a neural network, here the FANN Library [3], can be used to automatically detect facial expressions. By statistical evaluation of variance in the examined face images, it is possible to let the relation between countenance and facial expression be learned by a neural network. This knowledge can then be used to classify new, unseen images.

Since the model may only deform within given borders, the parameters describe always possible variations of face expression. Classification errors due to impossible forms are a priori excluded.

Relevant literature:


For those who can read German, there is also my Masterthesis:

- A. Arnold, Automatische Erkennung von Gesichtsausdrücken auf der Basis statistischer Methoden und neuronaler Netze. Masterthesis, University of Applied Sciences of Mannheim, Deutschland, February 2005

1.2 Requirements

This application was designed to run under a GNU/Linux environment. As it is using VXL and cmake it might be also be run under Windows, but without any guaranty of damage or missbehaviour. The following
list should contain all needed third-party software, but there might be some additional dependencies by used libraries, which are not listed.

Requirements:

- The Vision-something-Libraries 1.3.0
  http://vxl.sourceforge.net/
- Intel Open Source Computer Vision Library 0.9.7
  http://www.intel.com/technology/computing/opencv/
- CMake
  http://www.cmake.org/
- Fast Artificial Neural Network Library
  http://leenissen.dk/fann/

The following tools were used while developing this software and might be useful for further developing:

- Eclipse 3.1.0, an integrated development environment
  http://www.eclipse.org/
- CDT 3.0.0, the C/C++ Development Tools
  http://www.eclipse.org/cdt/
- Eclox 0.3.4, an doxygen plug-in for eclipse
  http://home.gna.org/eclox/
- Doxygen version 1.3.6-20040222
- gcc (GCC) 3.3.5 (Debian 1:3.3.5-12)

## 1.3 Installation

The project should be easily set up within eclipse by importing the source-folder into the workspace. As it uses cmake, the project must be Standard (not a Managed) C++ Project and there should be No Indexer assigned to the project.

By default, some of the installation scripts of the libraries will try to install the files in `/usr/local/bin` or `/usr/lib`. If you are not root, you can specify an installation prefix other than `/usr/local` by giving `/configure` the option `--prefix=PATH`. Also make sure, that you compile the VXL libraries with vgui support, which is not enabled by default!

A reasonable order would be:

1. Install CMake
2. Install/Compile the VXL libraries and run some examples!
3. Install/Compile the FANN libraries and try the example
4. Install/Compile the OpenCV libraries
5. Run FETA
1.4 Walkthrough

The process of classifying facial expressions within a given image with this application is divided in several steps. This includes setting up a suitable trainset, train the ASM/AAM/ANN and classify images against them. Each step needs the prior ones or a stored object, which can be read from the harddisk into memory. This application is designed to offer support for all steps of classification.

The main idea is, that the system is trained by observing several images, showing faces in a frontal perspective. Therefor for each image exists an shape-description file (asf), containing the coordinates of landmark-points. In addition, each image is classified within the smd-file, to represent a specific expression. These expression is given as a number, the last argument in each line of the smd-file, starting at zero.

After training of the statistical model and the neural network, testing an image will result in a vector of bits, 0s and 1s. The first digit is the lowest expression (labelled with ’1’), the second is the bit for the second expression... and so on.

As the application is using vgui, all the default vgui-commands can be used, some of the basics are:

- CTRL+LEFT : zoom in
- CTRL+RIGHT : zoom out
- CTRL+MIDDLE : move image within view
- ALT+RIGHT : resize window
- ALT+LEFT : move window
- CTRL+Q : quit application

There are also several shortcuts for each function in the menu.

1.4.1 File-Menu

The File-dialog offers several basic options:

Display shape-model-data (smd) - allows to show all images described by a smd-file. The images are loaded into 'decks', which allows them to be browsed by using the PAGE-UP and PAGE-DOWN keys.

Display point distribution - this shows the point distribution over a trainingset (after alignment). All landmark points are projected into one view, showing the conection topology and the meanshape as well.

Preferences - This option offers the possibility to presetup all paths, which can otherwise configured later during the several dialogs.

Quit application - the 'clean' way to quit the application

1.4.2 Learn-Menu

Enable drag-mode - This might be the most complicate part of the application. It is used to learn new shapes by setting up the landmark-points manually. After activating the drag’n’drop mode the curently displayed shape can be edit by moving the point with pressed-mouse-button. SHIFT+LEFT offers the possibility to move Pointgroups like eye or nose, with SHIFT+RIGHT the entire shape can be moved to a better position.

Save & Exit-mode - Saving the Shape to an asf-file will also quit the drag’n’drop mode, for exiting without saving, just open the save-dialog but press cancel instead of OK.
Start 'learn-batch' - The learn-batch is an automated learning procedure. If there exists a small training set (usually set up by hand) you can start a batch, processing all images within a folder by typing a location like `./\*/*.jpg` within the opendialog. It will also generate a smd-file which contains all generated shape-files. Make sure to edit manually the number of files in this description, because it's generated within an iterator and setting this number automatically would need a second for-loop or more variables in memory :)

1.4.3 Train-Menu

Train model - The training of the AAM will produce a aam-file, which can be later used to load the AAM into memory without the high computational costs for training, and a ASCII file, containing the Eigenvalues for each shape within the training set.

Train network - Similar to the AAM, the ANN is trained, using the Eigenvalues as input and writing an ASCII description file of the trained ANN.

1.4.4 Test

Test model - This function calculates the shape of an unseen face within a given image. The resulting Shape parameters (eigenvalues) are saved to disk.

Test network - This function reads the Eigenvalues (see above) and tries to classify the facial expression of the corresponding image/face. The result of this process is print out on shell and statusbar, using the same bitwise structure, that is used within the eval-files.

Start automated test - This will run the same test for several images in a folder (specified like `./\*/*.jpg`) and save a list with results (including the time needed to classify each image) into a file.

1.4.5 some more advise

As this is only a brief description, the best thing is to learn by using the application. (That means also to find some bugs by clicking around, where I have never thought you would click :) At the moment you should always quit and restart the application, if you want to train a new AAM or ANN.

to be continued
Chapter 2

Facial Expression Tracking Application (FETA) Data Structure Documentation

2.1 AAM Class Reference

#include <AAM.h>

Inheritance diagram for AAM:

Collaboration diagram for AAM:

2.1.1 Detailed Description

AAM : base class for active appearance model

Author:  
: Jason Saragih (2005)
This is the parent class for all flavours of implemented 2D AAM’s in this library. It contains all commonly used functions and common variables. Most functions are written with reference to the following papers:
(1) ”Training Models of Shape from Sets of Examples” by Cootes et al. (2) ”Active Appearance Models Revisited” by I. Matthews and S. Baker
For details of usage of the functions refer to ”AAM.cc”.

Public Member Functions

- int Train (int n, int L, int N, double sfrac, double afrac, const char **imFname, vgl_point_2d< double > **imPoint, vnl_matrix< int > *tri)

Protected Attributes

- int __L
  number of points in model
- AffineMesh *__mesh
  number of Gaussian Pyramid levels
- LinearModel *__shape
  piecewise affine mesh for every level
- LinearModel *__apper
  linear shape model for every level
- vnl_vector< double > *__norms
  linear appearance model for every level
- vnl_matrix< double > *__simil
  normalising coeffs
- vnl_vector_fixed< double, 4 > *__pose
  similarity transform bases
- vnl_vector< double > *__s2D
  current pose params parameters to speed up computation

2.1.2 Member Function Documentation

2.1.2.1 int Train (int n, int L, int N, double sfrac, double afrac, const char **imFname, vgl_point_2d< double > **imPoint, vnl_matrix< int > *tri)

added initialization, aa. 16.12.2005
Here is the call graph for this function:
The documentation for this class was generated from the following files:

- AAM.h
- AAM.cc
2.2  AAM_Facade Class Reference

#include <AAM_Facade.h>

Inheritance diagram for AAM_Facade:

Collaboration diagram for AAM_Facade:

2.2.1  Detailed Description

This class is a facade for the ‘AAM_POIC’ / ‘AAM_SIC’ classes. It provides all needed methods to work with the GUIManager, prescribed by the (interface like) abstract class 'IFacade'.

As the AAM classes are not developed within this project, the facade also implements some functions needed to adapt these classes for example to the dataholding within this project.

Therefore it use also static functions out of 'Shared’ module, which (as the name says) are shared with the ASM modules

See also:
IFacade
AAM_POIC

Note:
requires the adding of two getters to AAM_POIC.h: pGetR() and pGetError()

Author:
Arne Arnold

Date:
30.10.2005

Version:
9.1
2.2 AAM_Facade Class Reference

Public Member Functions

- **AAM_Facade ()**  
  initialise all class-attributes to zero

- **virtual ~AAM_Facade ()**  
  delete all dynamic class-attributes

- **int TrainModel (const char *strModelFile, const char **cvStrImgSrc, const char **cvStrAsfSrc, const char *cStrTriFile, const int cNumImgs, const int cNumPts, const int cNumLvl, const double cFrac, int cBufSize=512)**  
  builds the statistical (AAM) model and save it to disk

- **int TrainModel (const char *strModelFile, const char *cPath, const int cBufSize, const char *cStrTriFile, const int cNumPts, const int cNumLvl, const double cFrac)**  
  builds the statistical (AAM) model and save it to disk (wrapper)

- **int TestModel (vil_image_view<vxl_byte> img, vnl_vector<double> *shape, const char *cStrImgSrc)**  
  estimates the shape of an unseen image, using the (AAM) modeldata

- **int Load (const char *model, int cBufSize=512)**  
  loads the preprocessed modeldata into memory

- **int Save (const char *model)**  
  save the current calculated model to disk

- **void easy2DMesh (vgui_easy2D_tableau_new obj2D, vnl_vector<double> &shape, vnl_vector<int> path, int drwamode=1)**  
  draw the shape (the points and lines) into a easy2D_tableau

- **int writePCAParameters (vcl_ostream &ofs, const char **cvStrImgSrc, const int cNumImgs, vnl_vector<int> vExpr, const int cNumExpr, vil_image_view<vxl_byte> *img, vnl_vector<double> *shape, vnl_vector<double> *p, vgui_easy2D_tableau_sptr obj2D=NULL)**  
  save the shape-parameters (the Eigenvalues) to disk

- **bool IsInit ()**  
  returns the state of the model (if it is build or not)

- **int GetNumExpr ()**  
  returns the number of trained expressions

- **int GetPntGrp (int idx)**  
  returns the group index, the given point belongs to

- **vnl_vector<int> GetPntGrp ()**  
  returns the vector with group indeces for each landmarkpoint

- **int varyShape (vnl_vector<double> *shape, int idx, double mul)**  
  generates a new shape by varying the shape-parameters
Protected Member Functions

- void SetIsInit (bool val)
  
  sets the state of the model

- int * pGetNumImgs ()
  
  returns the pointer on number of images within training data

- int * pGetNumExpr ()
  
  returns pointer on number of trained expressions

- int * pGetNumTpls ()
  
  returns pointer on number of templates

- vnl_vector<int>** pGetPntGrp ()
  
  returns a pointer on the group-value within shape-model

- int ** pGetTplPts ()
  
  returns pointer on template corresponding model points

- vnl_vector<int>** pGetExpr ()
  
  returns pointer on trained expressions

- char *** pGetImgSrc ()
  
  returns pointer on trained image-paths

- char *** pGetAsfSrc ()
  
  returns pointer on paths to trained model description files

- char *** pGetTplSrc ()
  
  returns pointer on paths to templates

Private Member Functions

- int writePCAParameters (vcl_ostream &ofs, const char ***cvStrImgSrc, const int cNumImgs, vnl_vector<int>** vExpr, const int cNumExpr)
  
  save the shape-parameters (the Eigenvalues) to disk (wrapper)

Private Attributes

- bool isInit_
  
  state of the model in memory

- int numImgs_
  
  number of images in training set

- char *** vStrImgSrc_
2.2 AAM_Facade Class Reference

vector for image URIs

- char ** vStrAsfSrc_
  vector for ASF-files

- int numExpr_
  maximum number of different expressions

- vnl_vector< int > * vPntGrp_
  vector for groups of landmarks

- vnl_vector< int > * vExpr_
  vector holding shown expression for each image

- int numTpls_
  number of templates for pose finding

- int * vTplPts_
  vector of indeces from template-corresponding modelpoints

- char ** vStrTplSrc_
  vector holding templates (images)

2.2.2 Member Function Documentation

2.2.2.1 void easy2DMesh (vgui_easy2D_tableau_new obj2D, vnl_vector< double > & shape, vnl_vector< int > path, int drawmode = 1) [virtual]

Method provides support for drawing the mesh, using a vnl_vector<double> instead of a vgl_point_2d<double> shape.

Parameters:
  - obj2D : easy2D_tableau to draw on
  - shape : shape to draw
  - path : vector of pathnumbers for each point
  - drawmode : selection, how to draw the shape

Returns:
  -1 if fails, 0 otherwise

optional: draw shape (vertices)
then draw points and connecting lines.
Implements IFacade.
Here is the call graph for this function:
2.2.2.2 int Load (const char * model, int cBufSize = 512)  [virtual]

Method overwrites the parent-method and add support for loading of local attributes

**Note:**
the load-/save-function of this class is NOT compatible to the corresponding ASM implementation, as it uses plain ASCII files instead of binary files

**Parameters:**
- *model*: path to model file
- *cBufSize*: size of stringbuffer used for I/O operations

**Returns:**
-1 if fails, 0 otherwise

call parent-class function
add support for groups of landmarks
add support for templates
add support for advanced trainingset-description

Implements IFacade.

Here is the call graph for this function:

```
AAM_Facade::Load
  AffineMesh::NPix
  AAM_Facade::pGetAsfSrc
  AAM_Facade::pGetExpr
  AAM_Facade::pGetImgSrc
  AAM_Facade::pGetNumExpr
  AAM_Facade::pGetNumImgs
  AAM_Facade::pGetNumTpls
  AAM_Facade::pGetPntGrp
  AAM_POIC::pGetR
  AAM_Facade::pGetTplPts
  AAM_Facade::pGetTplSrc
  AAM_Facade::SetIsInit
```

2.2.2.3 int Save (const char * model)

Method overwrites the parent-method and add support for saving local attributes

**Note:**
the save-/load-function of this class is NOT compatible to the corresponding ASM implementation, as it uses plain ASCII files instead of binary files
Parameters:

\textit{model} : path to model file

Returns:

-1 if fails, 0 otherwise

call parent-class function
add support for groups of landmarks
add support for templates
add support for advanced trainingset-description
Reimplemented from \texttt{AAM\_POIC}.

Here is the call graph for this function:

\begin{center}
\includegraphics[width=\textwidth]{call_graph.png}
\end{center}

\subsection*{2.2.2.4 \texttt{int TestModel} \quad \texttt{(vil\_image\_view< vxl\_byte > \texttt{img}, vnl\_vector< double > * \texttt{shape}, const char * \texttt{cStrImgSrc}) \quad [virtual]}}

Method calculates the shape of the face, shown within the given image and return the original image and the estimated shape (seperated) for further processing.

Parameters:

\texttt{img} : image to search the face in (must be already contain the searchimage)
\texttt{shape} : vector to store the shape in
\texttt{cStrImgSrc} : location of testimage, as openCV doesn’t support \texttt{vil\_image\_view}

Returns:

-1 if fails, 0 otherwise

The FaceDetect-class is used, to find the initial poseparameters.
Another template-matching is computed to get the scale factor and a better position, using normalized cross correlation (ncc).
After this, the particular shape of the face in the image is estimated by the \texttt{AAM}.
Implements IFacade.

Here is the call graph for this function:

![Call Graph](call_graph.png)

### 2.2.2.5 int TrainModel (const char * `cStrModelFile`, const char * `cPath`, const int `cBufSize`, const char * `cStrTriFile`, const int `cNumPts`, const int `cNumLvl`, const double `cFrac`) [virtual]

Methods reads the Trainingdatadescription from the smd-file, builds the model in memory and finally write it to disk. (wrapper)

**Parameters:**

- **cStrModelFile**: file to save the Model
- **cPath**: Path to the SMD trainingsata file
- **cBufSize**: size of stringbuffer used for I/O operations
- **cStrTriFile**: path to triangle-mesh-desription
- **cNumPts**: number of points used in shape-model
- **cNumLvl**: number of resolution levels to use
- **cFrac**: level of convergence

**Returns:**

-1 if fails, 0 otherwise

Frst, the smd-file, containing the list of asf-files, images & corresponding expressions is proceeded. Then the statistical (AAM) model ist calculated / build out of the trainingdata.

Implements IFacade.

Here is the call graph for this function:
2.2.2.6 \textbf{int TrainModel (const char} ∗\texttt{cStrModelFile, const char} ∗\texttt{cvStrImgSrc, const char} ∗\texttt{cvStrAsfSrc, const char} ∗\texttt{cvStrTriFile, const int} cNumImgs, const int cNumPts, const int cNumLvl, const double cFrac, int cBufSize = 512)\textbf{)}

Methods builds the statistical model from the given training data and stores it in a file.

\textbf{Parameters:}
- \texttt{cStrModelFile} : file to save the AAModel
- \texttt{cvStrImgSrc} : array of image file-locations
- \texttt{cvStrAsfSrc} : array of model files (asf)
- \texttt{cNumImgs} : size of training set (number of images/model files)
- \texttt{cBufSize} : size of string buffer used for I/O operations
- \texttt{cvStrTriFile} : path to triangle mesh description
- \texttt{cNumPts} : number of points used in shape model
- \texttt{cNumLvl} : number of resolution levels to use
- \texttt{cFrac} : level of convergence

\textbf{Returns:}
- -1 if fails, 0 otherwise

for each image in training set ...
... read the landmark points from the shape description file (asf)

Then, read triangles out of the triangulation description file (tri).

Finally, calculate the statistical AAM model.

Here is the call graph for this function:
2.2.2.7 int varyShape (vnl_vector<double> *shape, int idx, double mul) [virtual]

Method is used by the GUIManager to visualize the mode-variation within the 3 times standard deviation

Parameters:
- shape: reference on shape to modify
- idx: the idx of the current modified parameter
- mul: the amount, by with the parameter should be modified

Returns:
-1 if fails, 0 otherwise

Implements IFacade.

2.2.2.8 int writePCAParameters (vcl_ostream &ofs, const char **cvStrImgSrc, const int cNumImgs, vnl_vector<int> vExpr, const int cNumExpr) [private, virtual]

Writes Eigenvalues for each given image/shape into filestream (wrapper)

Parameters:
- ofs: output filestream
- cvStrImgSrc: array of imagefile locations
- cNumImgs: total number of images (size of array)
- vExpr: vector of expressions within the images
- cNumExpr: number of different expressions (highest value within array)

Returns:
-1 if fails, 0 otherwise

Implements IFacade.

Here is the call graph for this function:
2.2.2.9  int writePCAParameters (vcl_ostream & ofs, const char ** cvStrImgSrc, const int cNumImgs, vnl_vector<int> vExpr, const int cNumExpr, vil_image_view<vxl_byte> * img, vnl_vector<double> * shape, vnl_vector<double> * p, vgui_easy2D_tableau_sptr obj2D = NULL) [virtual]

Method estimates (for each image within the given array) the shape and writes the PCA parameters (Eigenvalues) into the given filestream. Pointers for img, shape, parameters are used to adress the results of the last processed image. Using a one-sized-imagesrc-array, the function can be used to export the PCA parameters and display the estimated shapes within one step.

Parameters:
  
  ofs : output filestream
  
  cvStrImgSrc : array of imagefilelocations
  
  cNumImgs : total number of images (size of array)
  
  vExpr : vector of expressions within the images
  
  cNumExpr : number of different expressions (highest value within array)
  
  img : pointer on vil_image_view, used during estimation
  
  shape : pointer on vnl_vector<double> shape instance
  
  p : pointer on vnl_vector<double> holding pca paramters
  
  obj2D : optional ref on vgui_easy2D-obj, where to draw the boundingboxes of the FaceDetector

Returns:
  
  -1 if fails, 0 otherwise

the method first loads image from file,

then the FaceDetect-class is used, to find the headpose within the image.

A second face-detection is computed to get the scale factor and a better position by applying a template-search, using normalized cross correlation (ncc).

After this, the particular shape of the face in the image is estimated by the AAM ...

... and the PCA parameters (Eigenvalues) of this shape are calculated to be stored in a FANN compilant file.

The number of lines, modes and expressions is stored in the first line, ...

... followed by the PCA parameters (Eigenvalues) ...

... and a line containing the bitwise encoded expressions.

Implements IFacade.

Here is the call graph for this function:
The documentation for this class was generated from the following files:

- AAM_Facade.h
- AAM_Facade.cc
#include <AAM_POIC.h>

Inheritance diagram for AAM_POIC:

```
AAM_POIC
  AAM_Facade
  AAM
```

Collaboration diagram for AAM_POIC:

```
LinearModel
  __shape
  __apper
AAM_POIC
AAM
  AffineMesh
  __mesh
```

## 2.3.1 Detailed Description

AAM_POIC : project out inverse compositional active appearance model

**Author:**
Jason Saragih (2005)

**See also:**
AAM

The Project-out Inverse Compositional Algorithm for Active Appearance Models image alignment. Most functions are written with reference to the following papers: (1) "Active Appearance Models Revisited" by I. Matthews and S. Baker (2) "Lucas Kanade 20 yeart On: A Unifying Framework: Part 3" by I. Matthews, R. Gross and S. Baker (equation 42)

For details of usage of the functions refer to "AAM_POIC.cc".

### Protected Member Functions

- `vnl_matrix<double> ** pGetR ()`
  
  *added, a.a. 24.11.2005*
Private Attributes

- vnl_vector< double > * __error
  
  \textit{Gauss-Newton descent matrix.}

The documentation for this class was generated from the following files:

- AAM_POIC.h
- AAM_POIC.cc
2.4 AAM_SIC Class Reference

#include <AAM_SIC.h>

Inheritance diagram for AAM_SIC:

Collaboration diagram for AAM_SIC:

2.4.1 Detailed Description

AAM_SIC : simultaneous inverse compositional active appearance model

Author:
Jason Saragih (2005)

See also:
AAM

The Simultaneous Inverse Compositional Active Appearance Models image alignment algorithm. Most functions are written with reference to the following papers: (1) "Lucas Kanade 20 yeart On: A Unifying Framework: Part 3" by I. Matthews, R. Gross and S. Baker (2) "Active Appearance Models Revisited" by I. Matthews and S. Baker

For details of usage of the functions refer to "AAM_SIC.cc"

Private Attributes

- vnl_matrix< double > * __Jy
  x-warp Jacobians

- vnl_matrix< double > * __dM
  y-warp Jacobians

- vnl_matrix< double > ** __dA
gradient of mean appearance

- \textit{vnl\_matrix} < double > * \textit{sd}
  gradient of appearance bases

- \textit{vnl\_matrix} < double > * \textit{H}
  steepest descent images

- \textit{vnl\_matrix} < double > * \textit{R}
  gauss newton hessian

- \textit{vnl\_vector} < double > * \textit{error}
  Gauss-Newton descent matrix.

The documentation for this class was generated from the following files:

- AAM\_SIC.h
- AAM\_SIC.cc
2.5 AffineMesh Class Reference

#include <AffineMesh.h>

2.5.1 Detailed Description

AffineMesh : piecewise affine mesh warp

Author:
Jason Saragih (2005)

Affine Mesh Class A morphing model for piecewise-linearly transforming points from a base mesh to a destination mesh.


Note: 1. For a fixed src and varying dest, CalcTri() and CalcAlphaBeta() only needs to be done once, for every new dest assign points, CalcCoeff() and Warp(). 2. For greater online speed precalculate points inside mesh and their corresponding triangles using CalcInterior(). Setup takes longer though. Call WarpInterior() to get morphs of all pixels within interior. 3. To calculate which triangles every vertex belongs to, use FindVTri()

Public Member Functions

- AffineMesh ()

Data Fields

- vgl_point_2d< double > * _dst

Private Member Functions

- int PtTriPair (vgl_point_2d< double > p, int t)

Private Attributes

- int __nTri
- int __nPix
number of triangles in mesh

- `vgl_box_2d<int>__bb`
  number of pixels in interior of mesh

- `vnl_vector<int>__pixTri`
  bounding box in base mesh

- `vnl_matrix<int>__tri`
  triangles for every pixel in mesh

- `vnl_matrix<int>__vtri`
  triangle vertices indices

- `vgl_point_2d<int>*__pix`
  vertex vs tri

- `vnl_matrix<double>__a`
  pixels within interior of mesh

- `vnl_matrix<double>__alpha`
  affine transform coeff's for all triangles

- `vnl_matrix<double>__beta`
  matrix of (c,x,y) coeffs for alpha

The documentation for this class was generated from the following files:

- AffineMesh.h
- AffineMesh.cc
2.6 ANN_Facade Class Reference

#include <ANN_Facade.h>

2.6.1 Detailed Description

This class is a facade for the FANN library. It provides methods to train and test against a neural network.

Author:
Arne Arnold

Date:
30.10.2005

Version:
9

Public Member Functions

- ANN_Facade ()
  initialize the class-attributes to zero

- ANN_Facade (const char *cStrInAnn)
  initialize the class-attributes from pre-trained file

- virtual ~ANN_Facade ()
  delete fann-object

- int Load (const char *cStrInAnn)
  load pre-trained ANN from disk

- int Train (const char *cStrInEval, const char *cStrOutAnn)
  train ANN from Eigenvalues

- vnl_vector<double> Test (vcl_istream &fileReader)
  test given row of eigenvalues against the ANN

- vnl_vector<double> Test (const char *cStrInEval)
  wrapper for Test(vcl_ifstream fileReader)

- bool IsInit ()
  returns the state of the ANN (if it is trained or not)

Protected Member Functions

- int ReadNumNodes (const char *cStrInEval)
  wrapper for ReadNumNodes(vcl_istream &fileReader)
• int ReadNumNodes (vcl_istream &fileReader)
  read & save header information (first line)

• void SetIsInit (bool val)
  set state of the ANN

Private Attributes

• bool __isInit
  state of the ANN

• fann * __ann
  the FANN object/instance

• unsigned int __num_input
  number of input nodes (eigenvalues)

• unsigned int __num_output
  number of output nodes (expressions)

2.6.2 Constructor & Destructor Documentation

2.6.2.1 ANN_Facade (const char * cStrInAnn)

initialise the class-attributes and loads a given pre-trained ANN into memory

Parameters:
  cStrInAnn : path to ANN-file

Here is the call graph for this function:

2.6.3 Member Function Documentation

2.6.3.1 int Load (const char * cStrInAnn)

loads pre-trained ANN into memory

Parameters:
  cStrInAnn : path to ANN-file

Returns:
  -1 if fails, 0 otherwise

Here is the call graph for this function:
2.6.3.2  **int ReadNumNodes (vcl_istream & fileReader)**  [protected]

read the first line (header) of the input Eigenvalues-file, provided by the AAM_Facade::Export-PCAParameters() method and store the values in the corresponding class-attributes ∗

**Parameters:**
- *fileReader*: (file-) stream

**Returns:**
- -1 if fails, 0 otherwise

2.6.3.3  **int ReadNumNodes (const char * cStrInEval)**  [protected]

read the first line (header) of the input Eigenvalues-file, provided by the AAM_Facade::Export-PCAParameters() method and store the values in the corresponding class-attributes (wrapper)

**Parameters:**
- *cStrInEval*: path to input (eval-)file (multiple lines of PCA Params)

**Returns:**
- -1 if fails, 0 otherwise

2.6.3.4  **vnl_vector<double> Test (const char * cStrInEval)**

check the given Eigenvalues against the ANN and get (test) the approximated Estimation (wrapper)

**Parameters:**
- *cStrInEval*: path to input (eval-)file (single line of PCA Params)

**Returns:**
- vector with bits, representing the expressions found.

Here is the call graph for this function:

```
ANN_Facade::Test ANN_Facade::Test ANN_Facade::ReadNumNodes
```

2.6.3.5  **vnl_vector<double> Test (vcl_istream & fileReader)**

check the given Eigenvalues against the ANN and get (test) the approximated Estimation

**Parameters:**
- *fileReader*: path to input (eval-)file (single line of PCA Params)

**Returns:**
- vector with bits, representing the expressions found.
Initialise the fileReader and reads the 1st line of the input file.
Then the Eigenvalues are parsed and classified by the ANN.
Here is the call graph for this function:

2.6.3.6 int Train (const char * cStrInEval, const char * cStrOutAnn)

train the ANN with the given Eigenvalues & Expressions and store the trained ANN as binary ann-file.

Parameters:
  cStrInEval : path to input (eval-)file (multiple lines of PCA Params)
  cStrOutAnn : path for output (ann-)file

Returns:
  -1 if fails, 0 otherwise

Here is the call graph for this function:

The documentation for this class was generated from the following files:

- ANN_Facade.h
- ANN_Facade.cc
2.7 ASM Class Reference

#include <ASM.h>

Inheritance diagram for ASM:

![Inheritance Diagram]

Collaboration diagram for ASM:

![Collaboration Diagram]

2.7.1 Detailed Description

This class describes the core of the Active Shape Model (as described by Cootes et.al.) and contains all the stuff which deals directly with the estimation of a unknown shape. It is based on the underlying Point-Distribution-Model, provided as Reference by the parent (startup) class.

Author:
Arne Arnold

Date:
13.10.2005

Version:
9

Public Member Functions

- **ASM ()**
  
  *calls init-method*

- **ASM (PDM *ref)**
  
  *calls init-method and register PDM object*

- **virtual ~ASM ()**
  
  *delete all dynamic class-attributes*

- **void init ()**
  
  * initialise all class-attributes*
• void registerPDM (PDM *ref)
    Sets reference on the PDM object, provided by parent Startup class.

• int init_GvProfile (int gsize=cGvTrain_, int Dim=cDim_)
    initialise the greyvalue profile

• vnl_vector<double> calc_2DProfileNormal (vnl_vector<double> &vtrShape, int idx, int pts=cGvTrain_, int Dim=cDim_)
    calculates the point-normale in the idx-th point

• void estimate_2DShape (int itol, double ftol, vnl_vector<double> *vtrEstShape, vil_image_view<vxl_byte> &image, double transx=0.0, double transy=0.0, double scale=1.0, int gsize=cGvTrain_, int gsearch=cGvSearch_, int Dim=cDim_)
    estimates the shape of the current image

• void save_trainingdata (const char *filename)
    save all necessary attributes to a (binary) file

• void load_trainingdata (const char *filename)
    restore all necessary attributes from a (binary) file

• bool isInit ()
    value of the given position in description vector

Protected Member Functions

• void save_trainingdata (vsl_b_ofstream &bfs)
    save all necessary attributes to a binary output stream

• void load_trainingdata (vsl_b_ifstream &bfs)
    restore all necessary attributes from a binary input stream

Private Member Functions

• vnl_vector<double> calc_normalized2DGvProfile (vnl_vector<double> &vtrPN, const vil_image_view<vxl_byte> &img)
    true if ASM has been initialized gets the grey-values along the given profile normal

• vnl_matrix<double> getCovarianceGV (vnl_vector<double> &vtrGV)
    builds a covariance matrix of the given vector
2.7 ASM Class Reference

Private Attributes

- vnl_matrix<double> * gvprofile_
  
  def.val. for m points in grey-value search profile (2m+1)

- bool isInit_
  
  Matrix of mean-greylevels over trainingimages along point normal for each point.

Static Private Attributes

- const int cDim_ = 2
  
  Pointer/Reference on parent PDM Object.

- const int cN1t_
  
  dimensions which are used for the model

- const int cGvTrain_ = 2
  
  factor t, by that the profile-normal is stretched

- const int cGvSearch_ = 4
  
  def.val. for k points in grey-value training profile (2k+1)

2.7.2 Constructor & Destructor Documentation

2.7.2.1 ASM (PDM * ref)

calls init-method and register PDM object

Parameters:

- ref : reference on PDM-Object

Here is the call graph for this function:

```
ASM::ASM
ASM::init
ASM::registerPDM
```

2.7.3 Member Function Documentation

2.7.3.1 vnl_vector<double> calc_2DProfileNormal (vnl_vector<double> & vtrShape, int idx, int pts = cGvTrain_, int Dim = cDim_)

This method calculates the normal on the axis, which is defined by the two neighbours of the i-th point. Then a profile-vector for the 2k+1 grey values along this normal (k values in each direction plus the grey value of the i-th point itself) is build.

Parameters:

- vtrShape : the coordinate vector of a shape
idx : the index of the i-th point, the normal is calculated for

pts : the number of points in the normal (on each side of i: pts*k+1)

Dim : Dimension of Points (ex. 2)

Returns:
vector with coordinates along the profile normal

Here is the call graph for this function:

ASM::calc_2DProfileNormal

2.7.3.2 vnl_vector< double > calc_normalized2DGvProfile (vnl_vector< double > & vtrPN, const vil_image_view< vxl_byte > & img) [private]

This method returns the greyvalues along a given profile normal and store them as normalized values within a vnl_vector. The sample is normalized by dividing through the sum of absolute elements values - return is a scalar (double).

Parameters:
vtrPN : the profile normal (with the point coordinates)
img : reference to the image where to read the grey-values

Returns:
vector with normalized greyvalues along the profile normal

2.7.3.3 void estimate_2DShape (int itol, double ftol, vnl_vector< double > * vtrEstShape, vil_image_view< vxl_byte > & image, double transx = 0.0, double transy = 0.0, double scale = 1.0, int gsize = cGvTrain_, int gsvsearch = cGvSearch_, int Dim = cDim_)

estimates the shape of the face in the given image, starting with the meanshape of the trainingset, scaled and pre-transformed by the given pose-parameters.

Parameters:
itol : maximum number of iterations to attempt
ftol : termination condition for adjustment process
vtrEstShape : shape which is estimated during process
image : reference to the current displayed vil_image object
transx : initial offset to move mean into image
transy : initial offset to move mean into image
scale : initial scale factor, to fit mean to (face-)image
gsize : size of the greyvalue profile in trainingset
gsvsearch : size of the greyvalue profile during search
Dim : Dimension of Points (ex. 2)

See also:
"Active Shape Models" by Cootes et al.
First we initialise the shape parameters to zero.
Then the the model instance $x = \text{mean}(x) + \Phi \cdot b$ is generated.
For each shape-point $i$ we then ...
... calculate the greyvalue profile along the normal ...
... and perform a cross-correlation search to find best match.
All points are so moved to better positions along the profile normal
Next we must find the pose parameters which best map $x$ to $Y$.
We do so by invert the pose parameters ...
... and use them to project $Y$ into the model co-ordinate frame.
In addition we project $y$ into the tangent plane to $\text{mean}(x)$ by scaling by $1/(y \ast \text{mean}(x))$
Finally we update the model parameters to match $y$, ...
... apply constraints on $b$ ...
... and re-generate the model instance $y$.
Here is the call graph for this function:

```
2.7.3.4  vnl_matrix<double> getCovarianceGV (vnl_vector<double> & vtrGV)  [private]
```

builds a covariance matrix of the given vector

**Parameters:**

$vtrGV$ : vector of greyvalues

**Returns:**

the covariance matrix

```
2.7.3.5  int init_GvProfile (int gsize = cGvTrain_, int Dim = cDim_)
```

This method calculates the normalized greyvalue-vectors for each image in the trainingset and store them as a matrix in the global attribute gprofile_

---

Generated on Mon Feb 6 15:27:02 2006 for Facial Expression Tracking Application (FETA) by Doxygen
Parameters:

$gvsize$  size of profile, given as $2 \times gvsize + 1$ points along normal

$Dim$  Dimension of Points (ex. 2)

Returns:

-1, if PDM is not initialized

Here is the call graph for this function:

```
ASM::init_GvProfile
ASM::calc_2DProfileNormal
ASM::calc_normalized2DGvProfile
PDM::isInit
PDM::n_shapes
PDM::read_points
PDM::shape_desc
PDM::read_asf_points
PDM::read_pts_points
```

2.7.3.6  void load_trainingdata (vsl_b_ifstream & bfs)  [protected]

loads all attributes of a saved ASM into memory. As the ASM is based also on the PDM, the ASM includes the attributes of the PDM by calling the PDM method as well.

Note:

the save-/load-function of this class is NOT compatible to the corresponding AAM implementation, as it use binary instead of plain ASCII files.

Parameters:

- $bfs$ : a binary-stream to read from

Here is the call graph for this function:

```
ASM::load_trainingdata
PDM::load_trainingdata
```

2.7.3.7  void load_trainingdata (const char * filename)

loads trainingsetdata from binary-file (wrapper)

Parameters:

- $filename$ : filename of trainingset-binary

2.7.3.8  void registerPDM (PDM * ref)

Sets reference on the PDM object, provided by parent Startup class

Parameters:

- $ref$ : Reference to PDM Object
2.7.3.9  void save_trainingdata (vsl_b_ofstream & bfs)  [protected]

saves all relevant attributes of learned ASM to binary-stream. As the ASM is based also on the PDM, the
ASM includes the attributes of the PDM by calling the PDM method as well.

**Note:**
the save-/load-function of this class is NOT compatible to the corresponding AAM implementation, as it
use binary instead of plain ASCII files.

**Parameters:**
- `bfs` : a given binary-stream, where to append the data

Here is the call graph for this function:

![Call Graph](image)

2.7.3.10  void save_trainingdata (const char * filename)

method for saving all trainingsetdata to binary-file (wrapper)

**Parameters:**
- `filename` : filename of trainingset-binary

The documentation for this class was generated from the following files:

- ASM.h
- ASM.cc
2.8  **ASM_Facade Class Reference**

#include <ASM_Facade.h>

Inheritance diagram for ASM_Facade:

```plaintext
ASM
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
ASM Facade

Collaboration diagram for ASM_Facade:

```plaintext
PDM
refPDM_
ASM
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
ASM Facade

2.8.1  **Detailed Description**

d this class is a facade for the **ASM** classes. It provides all needed Methods to work with the **GUIManager**, prescribed by the (interface like) abstract class 'IFacade'.

It use also static functions out of 'Shared' modul, which (as the name says) are shared with the **ASM** modules

See also:

  IFacade
  ASM

Author:

  Arne Arnold

Date:

  10.01.2006

Version:

  9

Public Member Functions

- **ASM_Facade ()**
  
  initialise all class-attributes to zero

- virtual ~**ASM_Facade ()**
  
  delete all dynamic class-attributes
• int `TrainModel` (const char *cStrModelFile, const char *cPath, const int cBufSize, const char *cStrTriFile, const int cNumPts, const int cNumLvl, const double cFrac)

  *builds the statistical (ASM) model and save it to disk*

• int `TestModel` (vil_image_view<vxl_byte> img, vnl_vector<double> *shape, const char *cStrImgSrc)

  *estimates the shape of an unseen image, using the (ASM) modeldata*

• int `Load` (const char *model, int cBufSize=512)

  *loads the preprocessed modeldata into memory*

• int `Save` (const char *model)

  *save the current calculated model to disk*

• void `easy2DMesh` (vgui_easy2D_tableau_new obj2D, vnl_vector<double> &shape, vnl_vector<int> path, int drwamode=1)

  *draw the shape (the points and lines) into a easy2D_tableau*

• int `writePCAParameters` (vcl_ostream &ofs, const char **cvStrImgSrc, const int cNumImgs, vnl_vector<int> vExpr, const int cNumExpr, vil_image_view<vxl_byte> *img, vnl_vector<double> *p, vgui_easy2D_tableau_sptr obj2D=NULL)

  *save the shape-parameters (the Eigenvalues) to disk*

• bool `IsInit` ()

  *returns the state of the model (if it is build or not)*

• int `GetNumExpr` ()

  *returns the number of trained expressions*

• int `GetPntGrp` (int idx)

  *returns the group index, the given point belongs to*

• vnl_vector<int> `GetPntGrp` ()

  *returns the vector with group indices for each landmarkpoint*

• int `varyShape` (vnl_vector<double> *shape, int idx, double mul)

  *generates a new shape by varying the shape-parameters*

**Protected Member Functions**

• void `SetIsInit` (bool val)

  *sets the state of the model*

• int * `pGetNumImgs` ()

  *returns the pointer on number of images within training data*

• int * `pGetNumExpr` ()

  *returns pointer on number of trained expressions*
• int * pGetNumTpls ()
  returns pointer on number of Templates

• vnl_vector< int > ** pGetPntGrp ()
  returns a pointer on the group-value within shape-model

• int ** pGetTplPts ()
  returns pointer on template corresponding modelpoints

• vnl_vector< int > ** pGetExpr ()
  returns pointer on trained expressions

• char *** pGetImgSrc ()
  returns pointer on trained image-paths

• char *** pGetAsfSrc ()
  returns pointer on paths to trained modeldescription-files

• char *** pGetTplSrc ()
  returns pointer on paths to templates

Private Member Functions

• int writePCAParameters (vcl_ostream &ofs, const char ***cvStrImgSrc, const int cNumImgs, vnl_<
vnl_vector< int > vExpr, const int cNumExpr)
  save the shape-parameters (the Eigenvalues) to disk (wrapper)

Private Attributes

• bool isInit_
  state of the model in memory

• PDM * refPDM_
  reference on Point-Distribution-Model

• int numImgs_
  number of images in trainingset

• char ** vStrImgSrc_
  vector for image URI

• char ** vStrAsfSrc_
  vector for ASF-files

• int numExpr_
  maximum number of different expressions
• \texttt{vnl\_vector\<int\> \texttt{*vPntGrp\_}}
  \textit{vector for groups of landmarks}

• \texttt{vnl\_vector\<int\> \texttt{*vExpr\_}}
  \textit{vector holding shown expression for each image}

• \texttt{int numTpls\_}
  \textit{number of templates for pose finding}

• \texttt{int \texttt{*vTplPts\_}}
  \textit{vector of indeces from template-corresponding modelpoints}

• \texttt{char ** vStrTplSrc\_}
  \textit{vector holding templates (images)}

\textbf{Static Private Attributes}

• \texttt{const int citol\_ = 100}
  \textit{def. val. for maximum number of iterations to attempt}

• \texttt{const double cftol\_ = 0.9}
  \textit{def. val. for maximum number of iterations to attempt}

\subsection{2.8.2 Member Function Documentation}

\subsubsection{2.8.2.1 void easy2DMesh (vgui_easy2D\_tableau\_new \texttt{obj2D}, \texttt{vnl\_vector\<double\>} \texttt{& shape},
\texttt{vnl\_vector\<int\> \texttt{path}, int drawmode = 1}) [virtual]}

Method provides support for drawing the mesh, using a \texttt{vnl\_vector\<double\>} instead of a \texttt{vgl\_point\_-2d\<double\>} shape.

\textbf{Parameters:}

\texttt{obj2D} : easy2D\_tableau to draw on

\texttt{shape} : shape to draw

\texttt{path} : vector of pathnumbers for each point

\texttt{drawmode} : selection, how to draw the shape

\textbf{Returns:}

-1 if fails, 0 otherwise

draws points and connecting line

Implements \texttt{IFacade}. 

\footnotesize
Generated on Mon Feb 6 15:27:02 2006 for Facial Expression Tracking Application (FETA) by Doxygen
2.8.2.2  int Load (const char * model, int cBufSize = 512) [virtual]

Method overwrites the parent-method and add support for loading of local attributes

Note: the save-/load-function of this class is NOT compatible to the corresponding AAM implementation, as it use binary instead of plain ASCII files.

Parameters:
- model : path to model file
- cBufSize : size of stringbuffer used for I/O operations

Returns:
-1 if fails, 0 otherwise

Implements IFacade.

Here is the call graph for this function:

![Call Graph](image)

2.8.2.3  int Save (const char * model)

Method overwrites the parent-method and add support for saving local attributes

Note: the save-/load-function of this class is NOT compatible to the corresponding AAM implementation, as it use binary instead of plain ASCII files.

Parameters:
- model : path to model file

Returns:
-1 if fails, 0 otherwise
Here is the call graph for this function:

2.8.2.4 int TestModel (vil_image_view< vxl_byte > img, vnl_vector< double > * shape, const char* cStrImgSrc) [virtual]

Method calculates the shape of the face, shown within the given image and return the original image and the estimated shape (separated) for further processing.

Parameters:
- **img**: image to search the face in (must be already contain the searchimage)
- **shape**: vector to store the shape in
- **cStrImgSrc**: location of testimage, as openCV doesn’t support vil_image_view

Returns:
-1 if fails, 0 otherwise

The FaceDetect-class is used, to find the initial poseparameters.

Another template-matching is computed to get the scale factor and a better position, using normalized cross correlation (ncc).

After this, the particular shape of the face in the image is estimated by the ASM.

Implements IFacade.

Here is the call graph for this function:
2.8.2.5  

```c
int TrainModel (const char * cStrModelFile, const char * cPath, const int cBufSize, 
const char * cStrTriFile, const int cNumPts, const int cNumLvl, const double cFrac)
[virtual]
```

Methods builds the statistical model from the given Training data and stores it in a file

**Parameters:**
- `cStrModelFile`: file to save the Model to
- `cPath`: Path to the SMD training data file
- `cBufSize`: size of string buffer used for I/O operations
- `cStrTriFile`: path to triangle-mesh description
- `cNumPts`: number of points used in shape-model
- `cNumLvl`: number of resolution levels to use
- `cFrac`: level of convergence

**Returns:**
- `-1` if fails, `0` otherwise

get list of asf-files, images & corresponding expressions ...
... and build the PDM model and the ASM greyvalue-profile.

Implements IFacade.

Here is the call graph for this function:
2.8.2.6 int varyShape (vnl_vector<double> *shape, int idx, double mul) [virtual]

Method is used by the **GUIManager** to visualize the mode-variation within the 3 times standard deviation

**Parameters:**
- **shape**: reference on shape to modify
- **idx**: the idx of the current modified parameter
- **mul**: the amount, by with the parameter should be modified

**Returns:**
-1 if fails, 0 otherwise

Implements **IFacade**.

Here is the call graph for this function:

2.8.2.7 int writePCAParameters (vcl_ostream & ofs, const char ** cvStrImgSrc, const int cNumImgs, vnl_vector<int> vExpr, const int cNumExpr) [private, virtual]

Writes Eigenvalues for each given image/shape into filestream (wrapper)

**Parameters:**
- **ofs**: output filestream
- **cvStrImgSrc**: array of imagefilelocations
- **cNumImgs**: total number of images (size of array)
- **vExpr**: vector of Expressions within the images
- **cNumExpr**: number of different expressions (highest value within array)

**Returns:**
-1 if fails, 0 otherwise

Implements **IFacade**.

Here is the call graph for this function:

2.8.2.8 int writePCAParameters (vcl_ostream & ofs, const char ** cvStrImgSrc, const int cNumImgs, vnl_vector<int> > vExpr, const int cNumExpr, vil_image_view<vxl_byte> * img, vnl_vector<double> > * shape, vnl_vector<double> > * p, vgui_easy2D_tableau_sptr obj2D = NULL) [virtual]

Method estimates (for each image within the given array) the shape and writes the PCA parameters (Eigenvalues) into the given filestream. Pointers for img, shape, parameters are used to adress the results of the last processed image. Using a one-sized-imagesrc-array, the function can be used to export the PCA parameters and display the estimated shapes within one step.
Parameters:

- `ofs` : output filestream
- `cvStrImgSrc` : array of image file locations
- `cNumImgs` : total number of images (size of array)
- `vExpr` : vector of Expressions within the images
- `cNumExpr` : number of different expressions (highest value within array)
- `img` : pointer on vil_image_view, used during estimation
- `shape` : pointer on vnl_vector<double> shape instance
- `p` : pointer on vnl_vector<double> holding pca parameters
- `obj2D` : optional ref on vgui_easy2D-obj, where to draw the bounding boxes of the FaceDetector

Returns:
- -1 if fails, 0 otherwise

The method first loads image from file, then the FaceDetect-class is used, to find the head pose within the image.

A second face-detection is computed to get the scale factor and a better position by applying a template-search, using normalized cross correlation (ncc).

After this, the particular shape of the face in the image is estimated by the ASM ...

... and the PCA parameters (Eigenvalues) of this shape are calculated to be stored in a FANN compilant file.

The number of lines, modes and expressions is stored in the first line, ...

... followed by the PCA parameters (Eigenvalues) ...

... and a line containing the bitwise encoded expressions.

Implements IFacade.

Here is the call graph for this function:

![Call Graph](image)

The documentation for this class was generated from the following files:

- ASM_Facade.h
- ASM_Facade.cc
2.9 FaceDetect Class Reference

#include <FaceDetect.h>

2.9.1 Detailed Description

this class provides static methods for face detection. It is used by the ASM/AAM facades to initialize the poseparameters. For detecting the a face in the given image, the Haar classifier from the openCV Libraries are used. This class is, so far, the only class which uses openCV library in addition to vxl

Author:
Arne Arnold

Date:
30.11.2005

Version:
9

Static Public Member Functions

• int detectFace (const char *filename, double *centerX, double *centerY, double *height, double *width, vgui_easy2D_tableau_sptr obj2D=NULL)
  find the position of the whole face within image

• int detectTemplate (const char *imagefile, double *cx, double *cy, double *h, double *w, double *s, int numTpls, vnl_vector< double > shape, int *vTplPts, char **vStrTplSrc, vgui_easy2D_tableau_sptr obj2D=NULL)
  find the position of a given number of templates within image

Static Public Attributes

• const char * cFaceTmpl_ = "templates/haarcascade_frontalface_default.xml"
  location of openCV classifier cascade XML file

2.9.2 Member Function Documentation

2.9.2.1 int detectFace (const char * imagefile, double * cx, double * cy, double * h, double * w, vgui_easy2D_tableau_sptr obj2D = NULL) [static]

find the position of the whole face within image by using a Haar Classifier

Parameters:
  imagefile: path (ie. full filename) of image
  cx : x-coordinate of centerpoint
  cy : y-coordinate of centerpoint
  h : height of boundingbox
**w**: width of boundingbox

**obj2D**: optional ref to vgui_easy2D-obj to draw the boundingbox of the found face

**Returns:**
-1 if fails, 0 otherwise

load classifier cascade from XML file object, holding the openCV classifier cascade

get the sequence of face rectangles
1.2 = scale the cascade by 20% after each pass
2 = groups of 3 (2+1)
or more neighbor face rectangles are joined into a single "face”, smaller groups are rejected
CV_HAAR_-DO_CANNY_PRUNING = use Canny edge detector to reduce number of false alarms
cvSize = start from the minimum face size allowed by the particular classifier used for face-detection (holding the detected face)

2.9.2.2 int detectTemplate (const char * imagefile, double * cx, double * cy, double * h, double * w,
double * s, int numTpls, vnl_vector< double > * shape, int * vTplPts, char ** vStrTplSrc,
vgui_easy2D_tableau_spotr obj2D = NULL) [static]

find the position of a given number of templates within image by using the normalized cross-correlation.

**Parameters:**

- **imagefile**: path (ie. full filename) of image
- **cx**: x-coordinate of centerpoint
- **cy**: y-coordinate of centerpoint
- **h**: height of boundingbox
- **w**: width of boundingbox
- **s**: scalefactor
- **numTpls**: Number of Templates (size of vector)
- **shape**: a shape to compare the width/height and caculate the scale
- **vTplPts**: vector of point-indeces, which corresponds to the templates
- **vStrTplSrc**: vector with templates-paths
- **obj2D**: optional reference to vgui_easy2D object, to draw the boundingboxes of found matches

**Returns:**
-1 if fails, 0 otherwise

First check, if the prior detectFace-Method had returned suitable results...

..and if the so specified Region of Interest (ROI) is within the image, if not adjust the values accordingly.

If no face was detect, search within the entire image

Set centerpoint (return-value), comparing to the center of the template

The documentation for this class was generated from the following files:

- FaceDetect.h
- FaceDetect.cc
2.10 GUIManager Class Reference

#include <GUIManager.h>

Collaboration diagram for GUIManager:

2.10.1 Detailed Description

This class holds the methods used to process callbacks and event handling in a singleton class. It’s a sub-class of the top-level tableau and provide custom processing of events by defining its own ::handle method. The methods on the manager provide the implementation for the menu callbacks.

See also:
vgui_wrapper_tableau

Author:
Arne Arnold

Date:
30.11.2005

Version:
9

Public Member Functions

- void init ()
  initialise all class-attributes to zero

- void initDisplay ()
  initialize the vgui-display-cascade and startscreen

- ~GUIManager ()
  delete all dynamic class-attributes

- void registerModel (IFacade *ref)
  Set Reference on the model-object.

- void registerANN (ANN_Facade *ref)
  Set Reference on ANN-object.

- void quit ()
  Quit the application.
• int showConfigDialog ()
  
  Opens a dialog to edit several parameters at a central stage.

• int trainModel ()
  
  Build/train an ASM/AAM out of a given smd-file description.

• int trainANN ()
  
  Build/train an ANN from the given Eigenvalues-file.

• int testModel ()
  
  Estimate a shape from a given image.

• int testANN ()
  
  Classify a facial expression within a given image.

• int testAutomate ()
  
  Classify each image in a given folder.

• int SaveCurShape ()
  
  Saves the currently displayed shape to an asf-file.

• int LearnFromBatch ()
  
  Builds a new trainingset by labeling all images automatic.

• int showShapeModelData ()
  
  Shows all shapes & images described by a given smd-file.

• int showPointdistribution ()
  
  Shows the pointdistribution of the selected smd-file.

• int show_var_mean (int idxEval, double *mul)
  
  Draw shape with variation in the i’th Eigenvalue.

• virtual bool handle (vgui_event const &e)
  
  the event handler

• void SetDragMode (int val)
  
  Set state of drag’n’drop mode.

• void SetDragPnt (int val)
  
  Set the pointindex of the selected shape point.

• int GetDragPnt ()
  
  Get the pointindex of the selected shape point.

**Static Public Member Functions**

• GUIManager * instance ()
  
  insure only one instance is created
Private Member Functions

- **GUIManager ()**
- **int calculateCoG (vnl_vector< double > shape, double *tx, double *ty)**
  calculates the center of gravity of a given shape

- **int getPoint (int curDeck, double x, double y, double tol)**
  return pointindex, if there is a shapepoint within 'tol' pixel to the mouse position

- **bool handleDrag (int curDeck, int dragMode, double px, double py)**
  eventhandler during drag’n’drop (shape-modifications) mode

Private Attributes

- **IFacade * refModel_**
  Pointer/Reference on parent model.

- **ANN_Facade * refANN_**
  Pointer/Reference on parent ANN.

- **vcl_vector< vnl_vector< double > > vShape_**
  vector with shape for each deck

- **vcl_vector< vnl_vector< double > > vP_**
  vector of PCA-parameters for each deck

- **vcl_vector< vil_image_view< vxl_byte > > vimg_**
  The vil_image_view<T> represents a view of a multi-plane image of given type.

- **vcl_vector< vgui_image_tableau_new > vitab_**
  loads and displays an image.

- **vcl_vector< vgui_easy2D_tableau_new > ve2D_**
  displays two-dimensional geometric objects.

- **vcl_vector< vgui_text_tableau_new > vtxt_**
  displays text

- **vcl_vector< vgui_composite_tableau_new > vcomp_**
  holds many children all of whom receive events.

- **vcl_vector< vel_string > viloc_**
  the location of the displayed images

- **vgui_deck_tableau_new deck_**
  holds many child tableaux but only one receives events.

- **vgui_viewer2D_tableau_sptr view_**
  functions to view 2D objects (zoom, drag, center).
• vgui_shell_tableau_new shell_
  the top-object within vgui

• unsigned int iNumPts_
  Number of Points in Shape.

• unsigned int iNumLv1_
  Number of Multi-resolution Levels.

• unsigned int iBufSize_
  Size of string buffer for I/O operations.

• unsigned int iDragMode_
  state of drag’n’drop mode

• int iDragPnt_
  point index of currently selected point

• int selEval_
  Eigenvalue selected for animation.

• double mulEval_
  current multiplicator for Eigenvalue 0=0%, 1=100%

• double dFrac_
  level of convergence

• vcl_string sProgRoot_
  configured program-root

• vcl_string sTriFile_
  configured / last used tri-file

• vcl_string sSmdFile_
  configured / last used smd-file

• vcl_string sModelFile_
  configured / last used aam-file

• vcl_string sAnnFile_
  configured / last used ann-file

• vcl_string sImgIn_
  configured / last used input image

• vcl_string sImgOut_
  configured / last used output image

• vcl_string sEvalSet_
configured / last used eval-file for trainingdata

- vcl_string sEvalOne_
  configured / last used eval-file for testing

### Static Private Attributes

- GUIManager * instance_ = 0
  holds the singleton

- vcl_string regExprTri_ = "*.tri"
  usual fileextension for vertice-files

- vcl_string regExprSmd_ = "*.smd"
  usual fileextension for smd-files

- vcl_string regExprAam_ = "*.aam"
  usual fileextension for aam-files

- vcl_string regExprAsf_ = "*.asf"
  usual fileextension for asf-files

- vcl_string regExprAnn_ = "*.ann"
  usual fileextension for ann-files

- vcl_string regExprJpg_ = "*.jpg"
  usual fileextension for jpg-files

- vcl_string regExprEval_ = "*.eval"
  usual fileextension for eval-files

### 2.10.2 Constructor & Destructor Documentation

#### 2.10.2.1 GUIManager () [inline, private]

The constructor for this class is made private because the only way the class should be invoked is through the ::instance method.

See also:
  vgui_wrapper_tableau

### 2.10.3 Member Function Documentation

#### 2.10.3.1 bool handle (vgui_event const & e) [virtual]

Since the manager is a sub-class of a vgui_wrapper_tableau it inherits the ::handle method. This inheritance enables the manager to implement its own event processing and then pass unused events onto the manager’s tableau children.
Parameters:

\( e \): event object

Returns:

boolean than pass or pass not the event further

dragModes: \( 1 \): initialized state \( 2 \): move point (left_mouse only) \( 3 \): move obj (left_mouse+ALT) \( 4 \):
move entire shape (left_mouse+SHIFT)

Here is the call graph for this function:

![Call Graph]

2.10.3.2  **GUIManager** ∗ instance ()  [static]

This method returns a unique single instance of the basic_manager class. This design is called a singleton pattern and is used when the class must maintain a unique consistent state across applications accessing the class. This requirement frequently arises in event handling where global access to the same event process is required. In our example, All events are funneled through the same basic_manager instance.

Here is the call graph for this function:

![Call Graph]

2.10.3.3  **void quit ()**

A typical menu callback method. This manager can be accessed by many different applications that need to load an image. Thus, the image GUI management code doesn’t have to be continually rewritten.

2.10.3.4  **void registerANN (ANN_Facade ∗ ref)**

Set Reference on ANN-object, holded by parent (Startup) class

Parameters:

\( ref \): Reference to ANN instance

2.10.3.5  **void registerModel (IFacade ∗ ref)**

Set Reference on the model-object, holded by parent (Startup) class
Parameters:
  \textit{ref} : Reference to model instance

2.10.3.6 \textbf{int show\_var\_mean (int evalIdx, double * mul)}

Draw shape with variation in the i’th Eigenvalue within the suitable limits (standarderivation)

Parameters:
  \textit{idxEval} : Index of the Eigenvalue which is animated
  \textit{mul} : multiplicator for variation

Note:
  varying the Output of the ‘showPointdistribution()’ Method might cause a segmentation-fault, as the landmark-group-vector (vP_) wont be initialized if no real image is loaded

Returns:
  -1 if fails, 0 otherwise

Here is the call graph for this function:

The documentation for this class was generated from the following files:

  • GUIManager.h
  • GUIManager.cc
2.11 GUI Menue Class Reference

#include <GUIMenue.h>

2.11.1 Detailed Description

This menu class is defined to package up the static callback functions required in the vgui_menu assembly. Note that the menu callback functions are paired with methods on the manager. These menus can be exported to other GUI libraries so that the same menu functionality can be re-used. Keep in mind that under Windows special measures must be taken to export static items.

Author:
Arne Arnold

Date:
30.11.2005

Version:
9 based on the VXL Examples for building an application with vgui

Static Public Member Functions

• vgui_menu get_menu ()
  basic_menus definitions

• void show_smd_callback ()
  display all shapes within description-file

• void show_pd_callback ()
  display pointdistribution in a trainingset

• void quit_callback ()
  quit application

• void config_callback ()
  configure all parameters central

• void enable_drag_callback ()
  enter drag’n’drop mode

• void save_drag_callback ()
  leave drag’n’drop mode

• void learn_batch_callback ()
  learn shape-description for whole directory

• void train_aam_callback ()
  train aam from smd-file

• void train_ann_callback ()
2.11 GUIMenue Class Reference

train ann from eval-file

• void test_aam_callback ()
  estimate shape for jpf-file

• void test_ann_callback ()
  classify eigenvalues within eval-file

• void test_automatic_callback ()
  classify automated each image within a given folder

Private Member Functions

• GUIMenue ()
  create menue

2.11.2 Member Function Documentation

2.11.2.1 vgui_menu get_menu () [static]

GUIMenue definitions

Returns:
  menubar the staffed vgui_menu-object

Here is the call graph for this function:

The documentation for this class was generated from the following files:

• GUIMenue.h
• GUIMenu.cc


2.12 IFacade Class Reference

#include <IFacade.h>

Inheritance diagram for IFacade:

![Inheritance Diagram](image)

2.12.1 Detailed Description

this class is a abstract class, used as kind of an interface to describe the needed functionality to work with the GUIManager.

Each facade must implement this class, as otherwise it will not be able to be used by the GUIManager.

Author:
Arne Arnold

Date:
11.01.2006

Version:
9

Public Member Functions

- virtual int GetNumExpr ()=0
  
  Get number of trained Expressions.

- virtual void easy2DMesh (vgui_easy2D_tableau_new obj2D, vnl_vector< double > &shape, vnl_vector< int > path, int drwamode=1)=0
  
  Draw shape to easy2D_tableau.

- virtual int TrainModel (const char *cStrModelFile, const char *cPath, const int cBufSize, const char *cStrTriFile, const int cNumPts, const int cNumLvl, const double cFrac)=0
  
  Train the statistical model and save it to disk.

- virtual bool IsInit ()=0
  
  Get state of the Model in memory.

- virtual int Load (const char *model, int cBufSize=512)=0
  
  Load the model into memory (from disk).

- virtual int writePCAParameters (vcl_ostream &ofs, const char **cvStrImgSrc, const int cNumImgs, vnl_vector< int > vExpr, const int cNumExpr)=0
  
  Save Eigenvalues to disk (wrapper).
• virtual int writePCAParameters (vel_ostream &ofs, const char **cvStrImgSrc, const int cNumImgs, vnl_vector< int > vExpr, const int cNumExpr, vil_image_view< vxl_byte > *img, vnl_vector< double > *shape, vnl_vector< double > *p, vgui_easy2D_tableau_sptr obj2D=NULL)=0
  
  Save Eigenvalues to disk.

• virtual int TestModel (vil_image_view< vxl_byte > img, vnl_vector< double > *shape, const char *cStrImgSrc)=0
  
  Test image against the model (estimate the shape).

• virtual int GetPntGrp (int idx)=0
  
  Get the number of group, the given point (index) belongs to.

• virtual vnl_vector< int > GetPntGrp ()=0
  
  Get the vector with groups/objects of the landmarkpoints.

• virtual vnl_vector< int > **pGetExpr ()=0
  
  Get pointer on trained Expressions.

• virtual int * pGetNumImgs ()=0
  
  Get pointer on number of Images within trainingdata.

• virtual int * pGetNumExpr ()=0
  
  Get pointer on number of trained Expressions.

• virtual char ***pGetImgSrc ()=0
  
  Get pointer on trained Image-paths.

• virtual char ***pGetAsfSrc ()=0
  
  Get pointer on paths to trained modeldescription-files.

The documentation for this class was generated from the following file:

• IFacade.h
2.13 LinearModel Class Reference

#include <LinearModel.h>

2.13.1 Detailed Description

Linear Model
model defining a linear object class where values are a linear combination of basis vectors

Author:
Jason Saragih (2005)

Public Member Functions

• LinearModel ()
  bases (columns)

Data Fields

• int _k
  number of dimensions
• vnl_vector< double > _p
  number of bases
• vnl_vector< double > _v
  principle component parameters
• vnl_vector< double > _P
  vector of variance
• vnl_vector< double > _M
  current vector
• vnl_matrix< double > _B
  mean

The documentation for this class was generated from the following files:

• LinearModel.h
• LinearModel.cc
2.14 PDM Class Reference

#include <PDM.h>

2.14.1 Detailed Description

disable contain all stuff wich deals with the establishment of the Point Distribution Model.

Author:
Arne Arnold

Date:
11.10.2005

Version:
9 new build, based on the latest version of Jasons's alignment methods and own but recomputed methods from earlier approaches.

Public Member Functions

- **PDM ()**
  initialise all class-attributes to zero

- **virtual ~PDM ()**
  delete all dynamic class-attributes

- **void init_trainingset (const char *filename, int itol, double ftol)**
  initialize the trainingset from the given smd-file

- **int n_shapes ()**
  number of shapes in trainingset

- **int n_points ()**
  number of points in shape

- **vnl_vector< double > shape_pts (int idx)**
  vector with coordinates of i-th shape

- **vnl_vector< double > shape_mean ()**
  returns the mean shape of the trainingset

- **const char * shape_asf (int idx)**
  name of corresponding asf-file

- **const char * shape_jpg (int idx)**
  name of corresponding jpg-file

- **int shape_desc (int idx, int pos)**
  value of the given position in description vector
• **vnl_matrix< int > shape_desc ()**
  *description vector*

• **bool isInit ()**
  *gives information about trainingset has already be loaded*

• **double get_tx_mean ()**
  *mean of translation in x-direction of all shapes in trainingset*

• **double get_ty_mean ()**
  *mean of translation in y-direction of all shapes in trainingset*

• **double get_tx (int idx)**
  *the translation parameter of the i-th shape during the alignment process (in x-direction)*

• **double get_ty (int idx)**
  *the translation parameter of the i-th shape during the alignment process (in y-direction)*

• **vnl_vector< double > vtrEV als ()**
  *Eigenvalues of the trainingset.*

• **void Align2DShapes (double *scale, double *theta, vnl_vector< double > &vtrSrc, vnl_vector< double > &vtrRef, int Dim=cDim_)**
  *Align 2 translation normalised shapes by rotation and scaling.*

• **vnl_vector< double > getEigenValuesByShape (vnl_vector< double > &vtrShape)**
  *approximate a shape by using the given vector of Eigenvalues*

• **vnl_vector< double > getShapeByEigenValues (vnl_vector< double > &vtrEV alues)**
  *the vector of Eigenvalues for a particular shape is calculated*

• **void adjustEV alues (vnl_vector< double > *vtrEV al)**
  *adjust Eigenvalues within suitable limits*

• **void old_adjustEV alues (vnl_vector< double > *vtrEV al)**
  *alternative implementation for adjustment of Eigenvalues*

• **double vtrPointProduct (vnl_vector< double > v1, vnl_vector< double > v2)**
  *calculates the point-product of two vectors*

• **void SimilarityTransform (double scale, double theta, double xtrans, double ytrans, vnl_vector< double > *vtrShape, int Dim=cDim_)**
  *Perform similarity transform on shape.*

• **vnl_vector< double > read_points (const char *filename, int Dim=cDim_, vnl_matrix< int > *ts-Desc=new vnl_matrix< int >*(0, 0, 0), int *emot=new int)**
  *read landmark points from inputfile (wrapper method)*

• **void save_trainingdata (vsl_b_ofstream &bfs)**
  *saves all relevant attributes of learned PDM to binary-stream*
• **void** `load_trainingdata` (vsl_b_ifstream &bfs)
  loads all attributes of a saved PDM into memory

• **void** `export_Evals` (const char *filename)
  export the learned Eigenvalues to a FANN compilant file (wrapper)

• **void** `export_Evals` (const char *filename, vnl_vector< double > &data)
  export the Eigenvalues of a given shape to a FANN compilant file (wrapper)

• **void** `write_asf_points` (const char *filename, vnl_vector< double > vtrShape, vnl_matrix<int> &tsDesc, vcl_string imagename, int emot, int Dim)
  saves the given shape (landmark) to a asf-file on disk

• **void** `add2trainingset` (const char *filename, const char *imagename, int itol, double ftol)
  methods adds new shape "on the fly" to the trainingset and reinitialize the ASM-Modeldata.

**Private Member Functions**

• **int** `read_shape_desc` (const char *filename, vcl_vector< vcl_string > *asfs, vcl_vector< vcl_string > *jggs)
  read list of shape description files from inputfile

• **void** `Align2DShapeSet` (int itol, double ftol, vnl_matrix< double > mtxSrc, vnl_matrix< double > *mtxDst, vnl_vector< double > *vtrM, int Dim=cDim_)
  Align all shapes in set to a common coordinate.

• **void** `calculateEigenValues` (vnl_matrix< double > &data, vnl_vector< double > &mean, vnl_matrix< double > *evecs, vnl_vector< double > *evals, int *modes, double ftol, int Dim=cDim_)
  calculates the Eigenvalues & vectors of the covariance Matrix

• **vnl_vector< double >** `read_asf_points` (const char *filename, int Dim, vnl_matrix< int > *tsDesc, int *emot)
  read landmark points from asf-inputfile

• **vnl_vector< double >** `read_pts_points` (const char *filename, int Dim, vnl_matrix< int > *tsDesc, int *emot)
  read landmark points from pts-inputfile

• **void** `read_parts` (const char *filename, vnl_matrix< int > *tsDesc)
  read inner shape description from parts-file

• **void** `export_Evals` (const char *filename, vnl_matrix< double > &data, vnl_vector< int > &emots)
  export all Eigenvalues within of a given set of shape

• **void** `save_trainingdata` (const char *filename)
  saves all relevant attributes of learned PDM to (binary) file (wrapper method)
• void load_trainingdata (const char *filename)
  
  restore all necessary attributes from a file (wrapper method)

Private Attributes

• vnl_matrix< double > * trainingset_
  number of emotions (= #neurons; without undefined)

• vnl_matrix< double > * transTxTy_
  the matrix, holding all shapes of the training

• vnl_matrix< int > * trainingsetDesc_
  the matrix, holding all shapes of the training

• vnl_vector< int > * trainingsetEmot_
• vcl_vector< vcl_string > * trainingsetASFs_
  array with all related facial expressions

• vcl_vector< vcl_string > * trainingsetJPGs_
  array with all asf-filenames

• vnl_vector< double > * vtrMean_
  array with all coresponding jpg-files

• vnl_matrix< double > * mtxEVecs_
  Mean of trainingset.

• vnl_vector< double > * vtrEvals_
  Matrix with the Eigenvectors of the trainingset.

• int maxEvals_
  Vector with the Eigenvalues of the trainingset.

• int rows_
  number of modes needed to reach 98% convergence

• int cols_
  rows of trainingset = number of shapes

• int leye_
  cols of trainingset = number of points * Dim

• bool isInit_
  index of the point, describing the left eye in the model (for reference)
Static Private Attributes

• const int `cDim_` = 2
  point dimensions of used data

• const int `cDescDim_` = 4
  dimensions which are used for the model

• const int `cEmots_` = 7
  dimensions which are used for the description matrix

2.14.2 Member Function Documentation

2.14.2.1 void add2trainingset (const char∗ filename, const char∗ imagename, int itol, double ftol)

methods adds new shape "on the fly" to the trainingset and reinitialize the ASM-Modeldata. Used for
labeling images. Results will be not stored in the smd-file and lost after restarting the application.

Parameters:
  `filename` : asf-description-file
  `imagename` : corresponding image
  `itol` : maximum number of iterations to attempt
  `ftol` : termination condition for adjustment process

Method first resizes the trainingset to match the expected amount of entries.
Then, the given shape vector is added to the trainingset...
... and the whole trainingset is aligned and...
... the Eigenvectors & Eigenvalues are recalculated

Here is the call graph for this function:

2.14.2.2 void adjustEValues (vnl_vector< double >∗ vtrEval)

adjust Eigenvalues within suitable limits by scaling b, using a hyperellipsioid described by Cootes, (eq. 4.8)

Parameters:
  `vtrEval` : reference on vector, which should be adjusted
2.14.2.3 void Allign2DShapes (double * scale, double * theta, vnl_vector<double> & vtrSrc, vnl_vector<double> & vtrRef, int Dim = cDim_)

Align 2 translation normalised shapes by rotation and scaling

Parameters:
- **scale**: scaling from vtrSrc to vtrRef
- **theta**: rotation from vtrSrc to vtrRef
- **vtrSrc**: shape to align
- **vtrRef**: shape to align to
- **Dim**: Dimension of Points (ex. 2)

Author:
Jason Saragih, Arne Arnold

See also:
"Statistical models of Appearance for Computer Vision", by T.F. Cootes and C. J. Taylor, Appendix B.

2.14.2.4 void Allign2DShapeSet (int itol, double ftol, vnl_matrix<double> mtxSrc, vnl_matrix<double> * mtxDst, vnl_vector<double> * vtrM, int Dim = cDim_) [private]

Align all shapes in set to a common coordinate

See also:
"Statistical models of Appearance for Computer Vision", by T.F. Cootes and C. J. Taylor, Section 4.2

Parameters:
- **ftol**: termination condition
- **itol**: maximum number of iterations to attempt
- **mtxSrc**: original shapes
- **mtxDst**: contains aligned shapes on return
- **vtrM**: the mean (of the traingset) where alignement should be based on
- **Dim**: Dimension of Points (ex. 2)

Author:
Jason Saragih, Arne Arnold

First, all shapes are normalized in translation.
Then, Procrustes alignment is applied until converged. This includes:
- calculate alignment rotation and scaling
- perform alignment
- transform to tangent space of centroid
- update mean
normalizing mean in reference to the first shape (or fixed values) is required in each iteration (according to Cootes et.al, "Active Shape Models")

Here is the call graph for this function:
2.14.2.5 void calculateEigenValues (vnl_matrix<double> & data, vnl_vector<double> & mean, vnl_matrix<double> * evecs, vnl_vector<double> * evals, int * modes, double ftol, int Dim = cDim_) [private]

For each shape in the training set the deviation from the mean is calculated to set up the 2nx2n covariance matrix as described by T.Cootes Equation (9). Then the Eigenvalues of the covariance Matrix are computed by using the vnl_symmetric_eigensystem_compute() function.

Author:
Jason Saragih, Arne Arnold

Method performs SVD...
...and sorts eigenvectors corresponding to decreasing values of eigenvalues
get variance

2.14.2.6 void export_Evals (const char * filename, vnl_matrix<double> & data, vnl_vector<int> & emot) [private]

export all Eigenvalues within of a given set of shape file is FANN compilant

Parameters:
filename filename to save
data matrix with shapes (eigenvalues will be calculated)
emots reference to vector with corresponding expressions (used by FAAN)

Here is the call graph for this function:

2.14.2.7 void export_Evals (const char * filename, vnl_vector<double> & data)

export the Eigenvalues of a given shape to a FANN compilant file (wrapper)

Parameters:
filename filename to save
data vector with eigenvalues

Here is the call graph for this function:
2.14.2.8  void export_Evals (const char * filename)

export the learned Eigenvalues to a FANN compilant file (wrapper)

Parameters:

   \texttt{filename} : filename to save

2.14.2.9  double get_tx (int idx)

Parameters:

   \texttt{idx} : the i-th shape

Returns:

   the translation parameter of the i-th shape during the alignment process (in x-direction)

2.14.2.10 double get_ty (int idx)

Parameters:

   \texttt{idx} : the i-th shape

Returns:

   the translation parameter of the i-th shape during the alignment process (in y-direction)

2.14.2.11 vnl_vector<double> getEigenValuesByShape (vnl_vector<double> & vtrShape)

the vector of Eigenvalues for a particular shape is calculated

Parameters:

   \texttt{vtrShape} : a given Shape

Returns:

   Eigenvalues for the Shape

2.14.2.12 vnl_vector<double> getShapeByEigenValues (vnl_vector<double> & vtrEV alues)

approximate a shape by using the given vector of Eigenvalues e.g. The two dimensional data is approxi-
mated using a model with a single parameter b

Parameters:

   \texttt{vtrEV alues} : Vector of Eigenvalues

Returns:

   vector of approximated shape
2.14.2.13 void init_trainingset (const char * filename, int itol, double ftol)

initialize the trainingset from the given smd-file the syntax of the smd-file is like described and used by Tim Cootes, but it’s needed to add the number of shapes quite in the beginning of the file i.e. "n_shapes: 18" and the templates, if used.

Parameters:
- **filename**: file with shape description
- **ftol**: termination condition for alignment process
- **itol**: maximum number of iterations to attempt

fills trainingset out of asf-file-vectors

assumes that all shape-vectors have the same size!

aligns the trainingset and ...

... calculate Eigenvectors & Eigenvalues

Here is the call graph for this function:

```
<table>
<thead>
<tr>
<th>PDM::init_trainingset</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDM::Allign2DShapeSet</td>
</tr>
<tr>
<td>PDM::calculateEigenValues</td>
</tr>
<tr>
<td>PDM::read_points</td>
</tr>
<tr>
<td>PDM::read_shape_desc</td>
</tr>
<tr>
<td>PDM::Allign2DShapes</td>
</tr>
<tr>
<td>PDM::SimilarityTransform</td>
</tr>
<tr>
<td>PDM::read_asf_points</td>
</tr>
<tr>
<td>PDM::read_pts_points</td>
</tr>
</tbody>
</table>
```

2.14.2.14 void load_trainingdata (const char * filename) [private]

restore all necessary attributes from a file (wrapper method)

Parameters:
- **filename**: filename of trainingset-binary

Here is the call graph for this function:

```
<table>
<thead>
<tr>
<th>PDM::load_trainingdata</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDM::load_trainingdata</td>
</tr>
</tbody>
</table>
```

2.14.2.15 void load_trainingdata (vsl_b_ifstream & bfs)

loads all attributes of a saved PDM into memory

Parameters:
- **bfs**: a binary-stream to read from

2.14.2.16 void old_adjustEValues (vnl_vector< double > * vtrEVAl)

alternative implementation for adjustment of Eigenvalues using the standard deviation
Parameters:
  \textit{\textit{vtrEval}} : reference on vector, which should be adjusted

compare vector ‘b’ with standard deviation
scale vector
if still not adjusted, call method recursive

\textbf{2.14.2.17} \texttt{vnl\_vector\textless\ double \textgreater\ read\_asf\_points (const char \texttt{\ast} filename, int \texttt{Dim}, vnl\_matrix\textless\ int \texttt{\ast} tsDesc, int \texttt{\ast} emot)} [private]

read landmark points from asf-inputfile

Parameters:
  \textit{filename} : file with pointdefinitions
  \textit{Dim} : Dimension of Points (ex. 2)
  \textit{tsDesc} : Reference to Vector where to store trainingset description information
  \textit{emot} : Reference to Vector where to store expression, shown by the underlying image

Returns:
  vector of shape points (x1.y1,x2.y2,...,xn.yn)

x/y-value proceeded, as described in the IMM Face DB docu

\textbf{2.14.2.18} \texttt{void read\_parts (const char \texttt{\ast} filename, vnl\_matrix\textless\ int \texttt{\ast} tsDesc)} [private]

read inner shape description from parts-file

Parameters:
  \textit{filename} file with pointdefinitions
  \textit{tsDesc} Reference to Vector where to store trainingset description information

Returns:
  vector of shape points (x1.y1,x2.y2,...,xn.yn)

connects from
connects to
extract list of indices
tokenizer - split string with indices at ","

\textbf{2.14.2.19} \texttt{vnl\_vector\textless\ double \textgreater\ read\_points (const char \texttt{\ast} filename, int Dim = cDim\_, vnl\_matrix\textless\ int \texttt{\ast} tsDesc = new vnl\_matrix\textless\int\texttt{\ast}(0,0,0), int \texttt{\ast} emot = new int)}

read landmark points from inputfile (wrapper)

Parameters:
  \textit{filename} : file with pointdefinitions
  \textit{Dim} : Dimension of Points (ex. 2)
Facial Expression Tracking Application (FETA) Data Structure Documentation

**tsDesc**: Reference to Vector where to store trainingset description information  
**emot**: Reference where to store expression, shown by the underlying image

### Returns:
vector of shape points (x1,y1,x2,y2,...,xn,yn)

Here is the call graph for this function:

2.14.2.20 `vnl_vector<double> read_pts_points (const char* filename, int Dim, vnl_matrix<int>* tsDesc, int* emot)` [private]

read landmark points from pts-inputfile

**Parameters:**
- `filename`: file with point definitions
- `Dim`: Dimension of Points (ex. 2)
- `tsDesc`: Reference to Vector where to store trainingset description information
- `emot`: reference where to store expression, shown by the underlying image

**Returns:**
vector of shape points (x1,y1,x2,y2,...,xn,yn)

2.14.2.21 `int read_shape_desc (const char* filename, vcl_vector<vcl_string>* asfs, vcl_vector<vcl_string>* jpgs)` [private]

read list of shape description files from inputfile

**Parameters:**
- `filename`: file with shape description
- `asfs`: vector to store asf-filenames in
- `jpgs`: vector to store jpg-filenames in

**Returns:**
number of shapes in set

2.14.2.22 `void save_trainingdata (const char* filename)` [private]

saves all relevant attributes of learned PDM to (binary) file (wrapper method)

**Parameters:**
- `filename`: filename of trainingset-binary

Here is the call graph for this function:
2.14.2.23  void save_trainingdata (vsl_b_ofstream & bfs)

saves all relevant attributes of learned PDM to binary-stream

Parameters:
- bfs  a given binary-stream, where to append the data

2.14.2.24  const char * shape_asf (int idx)

Parameters:
- idx  index of shape

Returns:
- name of corresponding asf-file

2.14.2.25  int shape_desc (int idx, int pos)

positions in description vector: \ 0: path 1: type \ 2: connects from 3: connects to

Parameters:
- idx  index of shape
- pos  type of description

Returns:
- value of the given position in description vector

2.14.2.26  const char * shape_jpg (int idx)

Parameters:
- idx  index of shape

Returns:
- name of corresponding jpg-file

2.14.2.27  vnl_vector< double > shape_pts (int idx)

Parameters:
- idx  index of shape

Returns:
- vector with coordinates of i-th shape
2.14.2.28  void SimilarityTransform (double scale, double theta, double xtrans, double ytrans, 
            vnl_vector<double> *vtrShape, int Dim = cDim_)

Perform similarity transform on shape

**Parameters:**
- *scale* : scaling parameter
- *theta* : rotation
- *xtrans* : x translation
- *ytrans* : y translation
- *vtrShape* : shape to transform
- *Dim* : Dimension of Points (ex. 2)

**Author:**
Jason Saragih, Arne Arnold

2.14.2.29  double vtrPointProduct (vnl_vector<double> v1, vnl_vector<double> v2)

calculates the point-product of two vectores

**Parameters:**
- *v1* : first vector (x.transpose)
- *v2* : second vector (x)

**Returns:**
the scalar product

2.14.2.30  void write_asf_points (const char *filename, vnl_vector<double> vtrShape, 
            vnl_matrix<int> &tsDesc, vcl_string imagename, int emot, int Dim)

saves the given shape (landmark) to a asf-file on disk method is used during setting up a new trainingset, 
with unlabeled images.

**Parameters:**
- *filename* : filename to save shapemodel
- *vtrShape* : vector, containing the shape
- *tsDesc* : reference to global shape-description-vector
- *imagename* : filename of image, which belongs to the shape
- *emot* : index of expression, shown in the image
- *Dim* : Dimension of Points (ex. 2)

2.14.3  **Field Documentation**

2.14.3.1  vnl_matrix<int>* trainingsetDesc_ [private]

matrix, holding inner structure of each shape 1st column = path / object 2nd column = point type (not used 
yet) 3rd column = left neighbour (connects from) 4th column = right neighbour (connects to)
2.14.3.2  vnl_vector<int>* trainingsetEmot_  [private]

type of emotion, shown in picture (for easy export to ANN), e.g. 0: not used (unset) 1: neutral 2: happy 3: sad 4: afraid 5: angry 6: surprised 7: disgusted

The documentation for this class was generated from the following files:

- PDM.h
- PDM.cc
2.15 Shared Class Reference

#include <Shared.h>

2.15.1 Detailed Description

this class contains all the static methods, both of the facades, the ASM_Facade and the AAM_Facade, use.

Author:
Arne Arnold

Date:
11.10.2005

Version:
9

Static Public Member Functions

• void adjustCoordinates (int *x1, int *y1, const vil_image_view<vxl_byte> &img)
  
  check if coordinates within image boundary and adjust them to the min/max allowed values

• int GetPointsFromASFFile (const char *cPath, vnl_vector<double> *shape, vnl_vector<int> *path, int n_pts)
  
  Reads in 2D points from the points file and stores values in the shape vector (vnl_vector).

• int GetPointsFromASFFile (const char *cPath, vgl_point_2d<double> *shape, vnl_vector<int> *path, int points)
  
  Reads in 2D points from the points file and stores values in the shape vector (vgl_point_2d).

• int GetSrcFromSMDFile (const char *cPath, char ***vStrImgSrc, char ***vStrAsfSrc, int *numImgs, vnl_vector<int> *numExpr, int *numExpr, const int cBufSize, int *numTpls, char ***vTplSrc, int ***vTplPts)
  
  read shape-model-description file

• int write_asf_points (const char *filename, vnl_vector<double> &vtrShape, vnl_vector<int> &shapeDesc)
  
  save Shape (-model) to disk

• int ExportPCAParameters (const char *filename, IFacade *Modelref)
  
  Facade for writePCAParameters().

2.15.2 Member Function Documentation

2.15.2.1 int ExportPCAParameters (const char * filename, IFacade * Modelref) [static]

save all Eigenvalues to a ASCII file writes Eigenvalues for each shape within the trainingdata to disk (used to train the ANN, File can easily be read by the FANN-library-function)
2.15 Shared Class Reference

Parameters:

- **filename** : path (ie. full filename) for eval-file
- **Modelref** : reference on used Model (AAM/ASM) for further processing

Returns:

-1 if fails, 0 otherwise

Here is the call graph for this function:

```
Shared::ExportPCAParameters
IFacade::pGetExpr
IFacade::pGetImgSrc
IFacade::pGetNumExpr
IFacade::pGetNumImages
IFacade::writePCAParameters
```

### 2.15.2.2 int GetPointsFromASFFile (const char * cFile, vgl_point_2d< double > * shape,
                                 vnl_vector< int > * path, int points)  [static]

Reads in 2D points from the points file and stores values in the shape vector by concatenating the x and y locations of every point. The format of the points file is the same that comes along with the IMM_Face_DB. Method returns shape as type of vgl_point_2d< double >

Parameters:

- **cFile** : path (ie. full filename) of point file
- **shape** : vector containing concatenated (alternating) (x,y) point locations.
- **path** : vector containing the pathnumbr (object) for each point within the model.
- **points** : number of points in shape

Returns:

-1 if fails, 0 otherwise

### 2.15.2.3 int GetPointsFromASFFile (const char * cFile, vnl_vector< double > * shape,
                                      vnl_vector< int > * path, int n_pts)  [static]

Reads in 2D points from the points file and stores values in the shape vector by concatenating the x and y locations of every point. The format of the points file is the same that comes along with the IMM_Face_DB. Method returns shape as type of vnl_vector< double >

Parameters:

- **cFile** : path (ie. full filename) of point file
- **shape** : vector containing concatenated (alternating) (x,y) point locations.
- **path** : vector containing the pathnumbr (object) for each point within the model.
- **n_pts** : number of points in shape

Returns:

-1 if fails, 0 otherwise
2.15.2.4 int GetSrcFromSMDFile (const char *cFile, char ***vStrImgSrc, char ***vStrAsfSrc, int *numImgs, vnl_vector< int > **vExpr, int *numExpr, const int cBufSize, int *numTpls, char ***vStrTplSrc, int ***vTplPts) [static]

Reads list of asf-files, image-locations and related expressions for building the trainigdata. The format of the shape-model-description file is the same as used by Tim Cootes within his application.

Parameters:
- cFile : path (ie. full filename) of smd-file
- vStrImgSrc : vector for storing the location of trainingimages
- vStrAsfSrc : vector for storing the location of shape-description-files (asf)
- numImgs : variable for size (number of images) of trainingdata
- vExpr : vector for related expression for each image within trainingdata
- numExpr : variable for max number used to describe a expression
- cBufSize : size of char-buffer used to temporary store the file-paths
- numTpls : number of templates (size of array)
- vStrTplSrc : array with templates (-paths)
- vTplPts : corresponding point-indices in the shape-model

Note:
a blank line within the shapedescriptin, which is not comment-out by '#', or '//', will produce a segmentation-fault

Returns:
-1 if fails, 0 otherwise

2.15.2.5 int write_asf_points (const char *filename, vnl_vector< double > &vtrShape, vnl_vector< int > shapeDesc) [static]

write landmark points to asf-file writes coordinates withgin given shape to a asf-file (to disk)

Parameters:
- filename : path (ie. full filename) for asf-file
- vtrShape : point-ccordinates as shape-vector
- shapeDesc : vector with pathnumbers for each point

Returns:
-1 if fails, 0 otherwise

The documentation for this class was generated from the following files:

- Shared.h
- Shared.cc
#include <Stopwatch.h>

## Detailed Description

This class contains methods used for time measurement. It implements start-stop functions or can be used by creating an instance and destroying it, the time-of-live is printed out on the console.

**Author:**

Arne Arnold

**Date:**

09.12.2005

**Version:**

9

### Public Member Functions

- **Stopwatch** (char *of="total time for last activity:%g seconds", vcl_ostream *ops=&vcl_cout)
  
  Creates an element of the timer-class

- **~Stopwatch** ()
  
  The destructor stops the timer and prints the elapsed time, before it finally destroys itself

- **void Start** ()
  
  Start timer

- **double Stop** ()
  
  Stop the timer and return the elapsed time

- **double getTime** ()
  
  Returns the last measured time

- **int getTicks** ()
  
  Returns the last measured cpu-ticks

### Private Attributes

- **clock_t start**
  
  Record start-time

- **clock_t last**
  
  Record stop-time

- **char * outputformat**
  
  Store given outputformat
• vcl_ostream * outputstream
  store given outputstream

2.16.2 Constructor & Destructor Documentation

2.16.2.1 Stopwatch (char * of = "total time for last activity: %g seconds", vcl_ostream * ops = &vcl_cout)

the default constructor creates an element of the timer-class. it also sets the target and format for the final output and starts the timer imidiately.

Parameters:
  of : string, which specify the output-format for the elapsed time
  ops : reference on outputstream to use, then destroyed

2.16.3 Member Function Documentation

2.16.3.1 int getTicks ()

get cpu-ticks between last start and stop

Returns:
  the last measured cpu-ticks

2.16.3.2 double getTime ()

get time between last start and stop

Returns:
  the last measured time

2.16.3.3 double Stop ()

stop the timer

Returns:
  the elapsed time

Here is the call graph for this function:

The documentation for this class was generated from the following files:

• Stopwatch.h
• Stopwatch.cc
Index

AAM, 5
    Train, 6
AAM_Facade, 8
    easy2DMesh, 11
    Load, 11
    Save, 12
    TestModel, 13
    TrainModel, 14, 15
    varyShape, 16
    writePCAParameters, 16, 17
AAM_POIC, 19
AAM_SIC, 21
add2trainingset
    PDM, 64
adjustEV alues
    PDM, 64
AffineMesh, 23
Allign2DShapes
    PDM, 64
Allign2DShapeSet
    PDM, 65
ANN_Facade, 25
    ANN_Facade, 26
    Load, 26
    ReadNumNodes, 26, 27
    Test, 27
    Train, 28
ASM, 29
    ASM, 31
    calc_2DProfileNormal, 31
    calc_normalized2DGvProfile, 32
    estimate_2DShape, 32
    getCOvarianceGV, 33
    init_GvProfile, 33
    load_trainingdata, 34
    registerPDM, 34
    save_trainingdata, 34, 35
ASM_Facade, 36
    easy2DMesh, 39
    Load, 39
    Save, 40
    TestModel, 41
    TrainModel, 41
    varyShape, 42
    writePCAParameters, 43
calc_2DProfileNormal
    ASM, 31
calc_normalized2DGvProfile
    ASM, 32
calculateEigenValues
    PDM, 66
detectFace
    FaceDetect, 45
detectTemplate
    FaceDetect, 46
easy2DMesh
    AAM_Facade, 11
    ASM_Facade, 39
estimate_2DShape
    ASM, 32
export_Evals
    PDM, 66
ExportPCAParameters
    Shared, 74
    FaceDetect, 45
    FaceDetect
detectFace, 45
detectTemplate, 46
get_menu
    GUIMenue, 55
get_tx
    PDM, 67
get_ty
    PDM, 67
getCovarianceGV
    ASM, 33
getEigenValuesByShape
    PDM, 67
GetPointsFromASFFile
    Shared, 75
getShapeByEigenValues
    PDM, 67
GetSrcFromSMDFile
    Shared, 75
getTicks
    Stopwatch, 78
ggetTime
Stopwatch, 78
GUIManager, 47
    GUIManager, 51
    handle, 51
    instance, 52
    quit, 52
    registerANN, 52
    registerModel, 52
    show_var_mean, 53
GUIMenu, 54
    get_menu, 55
handle
    GUIManager, 51
IFacade, 57
init_GvProfile
    ASM, 33
init_trainingset
    PDM, 67
instance
    GUIManager, 52
LinearModel, 59
Load
    AAM_Facade, 11
    ANN_Facade, 26
    ASM_Facade, 39
load_trainingdata
    ASM, 34
    PDM, 68
old_adjustEV alues
    PDM, 68
PDM, 60
    add2trainingset, 64
    adjustEV alues, 64
    Align2DShapes, 64
    Align2DShapeSet, 65
    calculateEigenValues, 66
    export_Evals, 66
    get_tx, 67
    get_ty, 67
    getEigenValuesByShape, 67
    getShapeByEigenValues, 67
    init_trainingset, 67
    load_trainingdata, 68
    old_adjustEV alues, 68
    read_asf_points, 69
    read_parts, 69
    read_points, 69
    read_pts_points, 70
    read_shape_desc, 70
    save_trainingdata, 70
shape_asf, 71
shape_desc, 71
shape_jpg, 71
shape_pts, 71
SimilarityTransform, 71
trainingsetDesc_, 72
trainingsetEmot_, 72
write_asf_points, 72
quit
    GUIManager, 52
read_asf_points
    PDM, 69
read_parts
    PDM, 69
read_points
    PDM, 69
read_pts_points
    PDM, 70
read_shape_desc
    PDM, 70
ReadNumNodes
    ANN_Facade, 26, 27
    registerANN
    GUIManager, 52
    registerModel
    GUIManager, 52
    registerPDM
    ASM, 34
Save
    AAM_Facade, 12
    ASM_Facade, 40
    save_trainingdata
    ASM, 34, 35
    PDM, 70
shape_asf
    PDM, 71
shape_desc
    PDM, 71
shape_jpg
    PDM, 71
shape_pts
    PDM, 71
Shared, 74
    ExportPCAParameters, 74
    GetPointsFromASFFile, 75
    GetSrcFromSMDFile, 75
    write_asf_points, 76
show_var_mean
    GUIManager, 53
    SimilarityTransform
PDM, 71
Stop
  Stopwatch, 78
Stopwatch, 77
  getTicks, 78
  getTime, 78
Stop, 78
  Stopwatch, 78
Test
  ANN_Facade, 27
TestModel
  AAM_Facade, 13
  ASM_Facade, 41
Train
  AAM, 6
  ANN_Facade, 28
trainingsetDesc_
    PDM, 72
trainingsetEmot_
    PDM, 72
TrainModel
  AAM_Facade, 14, 15
  ASM_Facade, 41
varyShape
  AAM_Facade, 16
  ASM_Facade, 42
vtrPointProduct
  PDM, 72
write_asf_points
  PDM, 72
  Shared, 76
writePCAParameters
  AAM_Facade, 16, 17
  ASM_Facade, 43