GSoC2011SfM

0.1

Generated by Doxygen 1.7.4

Sat May 28 2011 17:38:54

Contents

Chapter 1

Class Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Class Documentation

3.1 OpencvSfM::Camera Class Reference

This class represent the physical device which take the pictures. It is not related to a 3D position which is the role of the class FieldOfView. The role of the class is to store only device related informations like intra parameters, radial and tangential distotion.

`#include <Camera.h>`

Public Member Functions

- **Camera** (cv::Mat intra_params=cv::Mat::eye(3, 3, CV_64F), cv::Vec< double, 6 > radial_dist=cv::Vec< double, 6 >(0.0, 0.0, 0.0, 0.0, 0.0, 0.0), cv::Vec< double, 2 > tangential_dist=cv::Vec< double, 2 >(0.0, 0.0))

Protected Attributes

- cv::Mat intra_params_

  store intra parameters(3∗3 matrix). This matrix contains focal informations, principal point coordinates and skew of axis
- cv::Mat inv_intra_params_

  This is the inverse transformation of intra_params_. Used to speed up calculus...
- cv::Vec< double, 6 > radial_dist_

  used to store radial dist parameters (/fk_1/f to /fk_6/f)
- cv::Vec< double, 2 > tangential_dist_

  used to store tangential dist parameters (/fp_1/f and /fp_2/f)
- unsigned char config_

  This attribut is used to know what we should estimate... If equal to 0, everything should be estimated...
- std::vector< FieldOfView > pointsOfView_

  vector of the differents positions of the camera. This will store images too.

3.1.1 Detailed Description

This class represent the physical device which take the pictures. It is not related to a 3D position which is the role of the class FieldOfView. The role of the class is to store only device related informations like intra parameters, radial and tangential distotion.

This class can be used to store device related informations like intra parameters, radial and tangential distortion. We use the so-called pinhole camera model. That is, a scene view is formed by projecting 3D points into the image plane using a perspective transformation. Usual notation says that a point [u,v] from an image is related to the point [X,Y,Z] using the following notation :

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

This leads to the following relation between local coordinates and global ones:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$

$$x' = x/z$$
$$y' = y/z$$

Additionnal radial and tangeancial distortion are modelized like this:

$$x'' = x' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + 2p_1 x' y' + p_2(r^2 + 2x'^2)$$

$$y'' = y' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + p_1(r^2 + 2y'^2) + 2p_2 x' y'$$

where $r^2 = x'^2 + y'^2$
$u = f_x * x'' + c_x$
$v = f_y * y'' + c_y$

radial_dist_ can be used to store /fk_1/f to /fk_6/f tangential_dist_ can be used to store /fp_1/f and /fp_2/f

Definition at line 48 of file Camera.h.

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

- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/Camera.h
- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/Camera.cpp

3.2 OpencvSfM::ExternPosEstim Class Reference

3.2.1 Detailed Description

Definition at line 5 of file ExternPosEstim.h.

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

- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/ExternPosEstim.h
- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/ExternPosEstim.cpp

3.3 OpencvSfM::FieldOfView Class Reference

This class represent the 3D position of the device which take the pictures. The role of the class is to store everything related to the filed of view: picture, 3D position, points, matches and 3D points.

`#include <FieldOfView.h>`

Public Member Functions

- FieldOfView ()
- FieldOfView (Camera *device, std::string imgFileName, cv::Mat rotation=cv::Mat::eye(3, 3, CV_64F), cv::Vec< double, 3 > translation=cv::Vec< double, 3 >(0.0, 0.0, 0.0))
- FieldOfView (Camera *device, cv::Mat image, cv::Mat rotation=cv::Mat::eye(3, 3, CV_64F), cv::Vec< double, 3 > translation=cv::Vec< double, 3 >(0.0, 0.0, 0.0))
- FieldOfView (Camera *device, PointsToTrack *points, PointsMatcher *matches=NULL, cv::Mat rotation=cv::Mat::eye(3, 3, CV_64F), cv::Vec< double, 3 > translation=cv::Vec< double, 3 >(0.0, 0.0, 0.0))
- FieldOfView (FieldOfView &otherFieldOfView)
- virtual ~FieldOfView (void)
- bool isRealFoV ()

Protected Attributes

- cv::Mat image_

  The picture we get from this position... Could be RGB, B&W...
- cv::Mat rotation_

  Rotation matrix R.
- cv::Vec< double, 3 > translation_

  Translation vector t.
- cv::Mat projection_matrix_

  redundancy but speed improvement
- Camera * device_

intra parameters and distortion coefs

- PointsToTrack * pointsInfo_

  Points which are found in the picture.

- PointsMatcher * matches_

  This object will store the differents matches estimations between this FoV and other related views... ! This object will be shared between different FoV !

- ExternPosEstim * **extern_position_**
- Points3DEstim * **points3D_**
- unsigned char config_

  This attribut is used to know what we should estimate... If equal to 0, everything should be estimated...

3.3.1 Detailed Description

This class represent the 3D position of the device which take the pictures. The role of the class is to store everything related to the filed of view: picture, 3D position, points, matches and 3D points.

We use the so-called pinhole camera model. That is, a scene view is formed by projecting 3D points into the image plane using a perspective transformation. Usual notation says that a point [u,v] from an image is related to the point [X,Y,Z] using the following notation :

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

This leads to the following relation between local coordinates and global ones:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$

$$x' = x/z$$
$$y' = y/z$$

Definition at line 47 of file FieldOfView.h.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 OpencvSfM::FieldOfView::FieldOfView ( )

If we use this contructor, the object is not correct! Indeed, we don't have the needed informations like the camera wich take this FoV. Use isRealFoV to test the validity of objects if needed.

Definition at line 12 of file FieldOfView.cpp.

3.3.2.2 OpencvSfM::FieldOfView::FieldOfView (Camera * *device*, std::string *imgFileName*, cv::Mat *rotation* = `cv::Mat::eye(3, 3, CV_64F)`, cv::Vec< double, 3 > *translation* = `cv::Vec< double, 3 >(0.0, 0.0, 0.0)`)

To create a field of view, we need two things : a camera, and an picture. Here we give an address of a Camera, and the file name of the picture. If we have more informations, we can use the last parameters...

Parameters

device	address of existing Camera. This camera can be calibrated or not...
imgFileName	file name of the image taken from the device...
rotation	Matrix of the known rotation (optional)...
translation	Vector of the known translation (optional)...

3.3.2.3 OpencvSfM::FieldOfView::FieldOfView (Camera * *device*, cv::Mat *image*, cv::Mat *rotation* = `cv::Mat::eye(3, 3, CV_64F)`, cv::Vec< double, 3 > *translation* = `cv::Vec< double, 3 >(0.0, 0.0, 0.0)`)

To create a field of view, we need two things : a camera, and an picture. Here we give an address of a Camera, and the picture (matrix). If we have more informations, we can use the last parameters...

Parameters

device	address of existing Camera. This camera can be calibrated or not...
image	Matrix of the image previously loaded...
rotation	Matrix of the known rotation (optional)...
translation	Vector of the known translation (optional)...

3.3.2.4 OpencvSfM::FieldOfView::FieldOfView (Camera * *device*, PointsToTrack * *points*, PointsMatcher * *matches* = `NULL`, cv::Mat *rotation* = `cv::Mat::eye(3, 3, CV_64F)`, cv::Vec< double, 3 > *translation* = `cv::Vec< double, 3 >(0.0, 0.0, 0.0)`)

To create a field of view, we need two things : a camera, and an picture. Somethinmes, we don't have the picture but only points: we can create the field of view using this constructor... If we have more informations, we can use the last parameters...

Parameters

device	address of existing Camera. This camera can be calibrated or not...
points	points to track and optionally the associated features...
matches	object where matches between our points and other FoV's points (optional)
rotation	Matrix of the known rotation (optional)...
translation	Vector of the known translation (optional)...

3.3.2.5 OpencvSfM::FieldOfView::FieldOfView (FieldOfView & *otherFieldOfView*)

This constructor will not copy values... This allow smart pointer behavior

Definition at line 84 of file FieldOfView.cpp.

3.3.2.6 OpencvSfM::FieldOfView::~FieldOfView (void) [virtual]

Destructor of FieldOfView, release all vectors... TODO: define how we should release the vectors...

Definition at line 88 of file FieldOfView.cpp.

3.3.3 Member Function Documentation

3.3.3.1 bool OpencvSfM::FieldOfView::isRealFoV ( ) [inline]

Use isRealFoV to test the validity of objects if needed.

Returns

true is the object is correctly loaded, false if you should not use it !

Definition at line 112 of file FieldOfView.h.

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

- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/FieldOfView.h
- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/FieldOfView.cpp

3.4 OpencvSfM::MotionProcessor Class Reference

This class try to create a commun interface for files loading. Indeed, if you want to use webcam, avi file of list of files, you will have to do some annoying processing, like iterate the different files of the directory. With MotionProcessor, you can now use a folder of image the same way you use a webcam or a video file.

`#include <MotionProcessor.h>`

Public Member Functions

- bool setInputSource (int idWebCam)
- bool setInputSource (std::string nameOfFile, TypeOfMotionProcessor inputType=IS_SINGLE_FILE)
- bool setInputSource (std::string prefix, std::string suffix, int startNumber=0)
- cv::Mat getFrame ()
- bool setProperty (int idProp, double value)
- double getProperty (int idProp)

Protected Attributes

- TypeOfMotionProcessor type_of_input_

  This attribut is used to know which type is the input (webcam, video file, list of file or just one image)
- cv::VideoCapture capture_

  When the camera is attached to an avi file or webcam, this will be usefull to get frame...
- std::vector< std::string > nameOfFiles_

  If the motion processor use directory as input, we store here the names of files.
- std::string sourceName_
- std::string suffix_
- unsigned int numFrame_

  When the camera is attached to a list of file, numFrame_ will be used to know how many frames we have take.
- int wantedWidth_

  if below 0, represent the wanted width of Mat returned by getFrame();
- int wantedHeight_

  if below 0, represent the wanted height of Mat returned by getFrame();
- bool convertToRGB_

  Boolean flags indicating whether images should be converted to RGB.

3.4.1 Detailed Description

This class try to create a commun interface for files loading. Indeed, if you want to use webcam, avi file of list of files, you will have to do some annoying processing, like iterate the different files of the directory. With MotionProcessor, you can now use a folder of image the same way you use a webcam or a video file.

The class is still in development as the way to open folder is not really clear... The easy way would be to use dirent.h header, but the easy thing is not always the best thing...

Definition at line 30 of file MotionProcessor.h.

3.4.2 Member Function Documentation

3.4.2.1 Mat OpencvSfM::MotionProcessor::getFrame ( )

use this method if you want to get a field of view from this camera. Be carreful : this function can return a fake FieldOfView! Indeed, if we are at the end of the video, we can't get an other frame...

Parameters

numFrame	if you want to get a previously loaded frame, you can use this param. If below 0, get a new frame...

Returns

a FoV. If the video is finished, the FoV returned is not usable! Test the validity with the bool FieldOfView::isRealFoV() method.

Definition at line 102 of file MotionProcessor.cpp.

3.4.2.2 double OpencvSfM::MotionProcessor::getProperty (int *idProp*)

use this method to get actual properties of pictures retrived by this MotionProcessor. the properties are the same than VideoCapture (see http://opencv.willowgarage.com/documentatio and_writing_images_and_video.html#cv-videocapture-get)

Parameters

idProp	Property identifier

Returns

the value of the property

Definition at line 270 of file MotionProcessor.cpp.

3.4.2.3 bool OpencvSfM::MotionProcessor::setInputSource (int *idWebCam*)

You can attach this camera to a webcam use this method to set it as the input source!

Parameters

idWebCam	id of the webcam

Returns

true if input source opened without problems

Definition at line 43 of file MotionProcessor.cpp.

3.4.2.4 bool OpencvSfM::MotionProcessor::setInputSource (std::string *nameOfFile*, TypeOfMotionProcessor *inputType* = IS_SINGLE_FILE)

You can attach this camera to a video file or a single picture. use this method to set it as the input source!

Parameters

nameOfFile	name of the media file (picture or avi movie)
type	of input (can be either IS_DIRECTORY, IS_VIDEO or IS_SINGLE_FILE)

Returns

true if input source opened without problems

3.4.2.5 bool OpencvSfM::MotionProcessor::setInputSource (std::string *prefix*, std::string *suffix*, int *startNumber* = 0)

You can attach this camera to a list of file. use this method to set the input source! For example, if the files are img1.jpg, img2.jpg, ... img125.jpg, prefix will be equal to "img", suffix to ".jpg" and startNumber equal to 1

Parameters

prefix	the part of the files names which stay the same (img)
suffix	the type of the files (.jpg for instance)
startNumber	the first number to use...

Returns

true if input source opened without problems

3.4.2.6 bool OpencvSfM::MotionProcessor::setProperty (int *idProp*, double *value*)

use this method to change the properties of pictures retrived by this MotionProcessor.
the properties are the same than VideoCapture (see http://opencv.willowgarage.com/documentation/cpp/and_writing_images_and_video.html#cv-videocapture-get)

Parameters

idProp	Property identifier
value	new value of the property

Definition at line 197 of file MotionProcessor.cpp.

3.4.3 Member Data Documentation

3.4.3.1 std::string OpencvSfM::MotionProcessor::sourceName_ [protected]

When the camera is attached to a list of file, sourceName_ will be used to store name of the prefix. For example, if the files are img1.jpg, img2.jpg, ... img125.jpg, sourceName_ will be equal to img

Definition at line 40 of file MotionProcessor.h.

3.4.3.2 std::string OpencvSfM::MotionProcessor::suffix_ [protected]

When the camera is attached to a list of file, suffix_ will be used to store name of the suffix. For example, if the files are img1.jpg, img2.jpg, ... img125.jpg, suffix_ will be equal to .jpg

Definition at line 45 of file MotionProcessor.h.

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

- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/MotionProcessor.h
- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/MotionProcessor.cpp

3.5 OpencvSfM::Points3DEstim Class Reference

3.5.1 Detailed Description

Definition at line 5 of file Points3DEstim.h.

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

- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/Points3DEstim.h
- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/Points3DEstim.cpp

3.6 OpencvSfM::PointsMatcher Class Reference

`#include <PointsMatcher.h>`

Public Member Functions

- **PointsMatcher** (const cv::Ptr< cv::DescriptorMatcher > &matcher)
- virtual void **add** (cv::Ptr< PointsToTrack > pointCollection)
- virtual void **clear** ()
- virtual void **train** ()
- virtual bool **isMaskSupported** ()
- virtual bool **empty** () const
- virtual cv::Ptr< PointsMatcher > **clone** (bool emptyTrainData=false) const
- virtual void **knnMatch** (cv::Ptr< PointsToTrack > queryPoints, std::vector< std::vector< cv::DMatch > > &matches, int k, const std::vector< cv::Mat > &masks, bool compactResult)
- virtual void **radiusMatch** (cv::Ptr< PointsToTrack > queryPoints, std::vector< std::vector< cv::DMatch > > &matches, float maxDistance, const std::vector< cv::Mat > &masks, bool compactResult)

Protected Attributes

- cv::Ptr< cv::DescriptorMatcher > **matcher_**
- std::vector< cv::Ptr< PointsToTrack > > **pointCollection_**

3.6.1 Detailed Description

A class used for matching descriptors that can be described as vectors in a finite-dimensional space

Definition at line 13 of file PointsMatcher.h.

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

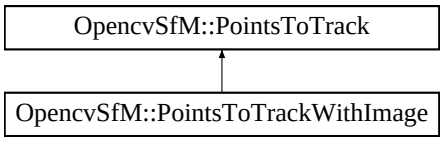
- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/PointsMatcher.h
- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/PointsMatcher.cpp

3.7 OpencvSfM::PointsToTrack Class Reference

This class can be used to store informations about points and features. This is an abstract class: you can't use it directly. Use for instance PointsToTrackSIFT.

`#include <PointsToTrack.h>`

Inheritance diagram for OpencvSfM::PointsToTrack:



Public Member Functions

- PointsToTrack (std::vector< cv::KeyPoint > keypoints=std::vector< cv::KeyPoint >(0), cv::Mat descriptors=cv::Mat())
- virtual ~PointsToTrack (void)
- virtual int computeKeypointsAndDesc ()
- virtual int computeKeypoints ()
- virtual void computeDescriptors ()
- virtual void addKeypoints (std::vector< cv::KeyPoint > keypoints, cv::Mat descriptors=cv::Mat(), bool computeMissingDescriptor=false)
- std::vector< cv::KeyPoint > getKeypoints () const
- cv::Mat getDescriptors () const
- void printPointsOnImage (const cv::Mat &image, cv::Mat &outImg, const cv::Scalar &color=cv::Scalar::all(-1), int flags=cv::DrawMatchesFlags::DEFAULT) const
- void **read** (const cv::FileNode &fn)
- void **write** (cv::FileStorage &fs) const

Protected Attributes

- std::vector< cv::KeyPoint > keypoints_

  This attribute will store points coordinates and sometimes orientation and size.

- cv::Mat descriptors_

  this attribute will store descritors for each points in a matrix with size (n∗m), where n is the number of points and m is the desciptor size.

3.7.1 Detailed Description

This class can be used to store informations about points and features. This is an abstract class: you can't use it directly. Use for instance PointsToTrackSIFT.

To create a structure from motion, most methods use points to compute the structure. This class focus on the first task in SfM: find points in image which are easy to track... When available, a feature vector for each points is very helpful: the matching will be easier.

Definition at line 18 of file PointsToTrack.h.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 OpencvSfM::PointsToTrack::PointsToTrack (std::vector< cv::KeyPoint > *keypoints* = `std::vector<cv::KeyPoint>(0)`, cv::Mat *descriptors* = `cv::Mat()`)

this constructor create an object with available information...

Parameters

keypoints	the points we will try to track...
descriptors	the feature vector for each points...

Definition at line 12 of file PointsToTrack.cpp.

3.7.2.2 OpencvSfM::PointsToTrack::~PointsToTrack (void) `[virtual]`

Destructor : delete points and features vectors

Definition at line 17 of file PointsToTrack.cpp.

3.7.3 Member Function Documentation

3.7.3.1 void OpencvSfM::PointsToTrack::addKeypoints (std::vector< cv::KeyPoint > *keypoints*, cv::Mat *descriptors* = `cv::Mat()`, bool *computeMissingDescriptor* = `false`) `[virtual]`

This method is used to add Keypoints...

Parameters

keypoints	Keypoints to add
descriptors	of points, if any
compute-MissingDe-scriptor	if true, the missing descriptors are computed.

Definition at line 41 of file PointsToTrack.cpp.

3.7.3.2 void OpencvSfM::PointsToTrack::computeDescriptors ( ) [virtual]

This method is used to compute only descriptors...

Reimplemented in OpencvSfM::PointsToTrackWithImage.

Definition at line 37 of file PointsToTrack.cpp.

Referenced by addKeypoints(), and computeKeypointsAndDesc().

3.7.3.3 int OpencvSfM::PointsToTrack::computeKeypoints ( ) [virtual]

This method is used to compute only Keypoints...

Returns

the number of points

Reimplemented in OpencvSfM::PointsToTrackWithImage.

Definition at line 31 of file PointsToTrack.cpp.

Referenced by computeKeypointsAndDesc().

3.7.3.4 int OpencvSfM::PointsToTrack::computeKeypointsAndDesc ( ) [virtual]

This method is used to compute both Keypoints and descriptors...

Returns

the number of points

Definition at line 23 of file PointsToTrack.cpp.

3.7.3.5 cv::Mat OpencvSfM::PointsToTrack::getDescriptors ( ) const [inline]

this method return the descritors for each points in a matrix with size (n∗m), where n is the number of points and m is the desciptor size.

Returns

descritors for each points in a matrix with size (n∗m), where n is the number of points and m is the desciptor size.

Definition at line 65 of file PointsToTrack.h.

3.7.3.6 std::vector<cv::KeyPoint> OpencvSfM::PointsToTrack::getKeypoints ( ) const [inline]

this method return the points coordinates and sometimes orientation and size

Returns

points coordinates and sometimes orientation and size

Definition at line 60 of file PointsToTrack.h.

3.7.3.7 void OpencvSfM::PointsToTrack::printPointsOnImage (const cv::Mat & *image*, cv::Mat & *outImg*, const cv::Scalar & *color* = `cv::Scalar::all(-1)`, int *flags* = `cv::DrawMatchesFlags::DEFAULT`) const

To show the points on image, use this function to draw points on it.

Parameters

image	Source image.
outImg	Output image. Its content depends on flags value what is drawn in output image. See possible flags bit values.
color	Color of keypoints
flags	Possible flags bit values is defined by DrawMatchesFlags (see http://opencv.willowgarage.com/documentation/cpp/features2d_-drawing_function_of_keypoints_and_-matches.html#cv-drawmatches)

Definition at line 64 of file PointsToTrack.cpp.

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

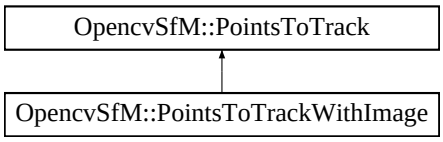
- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/PointsToTrack.h
- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/PointsToTrack.cpp

3.8 OpencvSfM::PointsToTrackWithImage Class Reference

This class can be used to find points and features in pictures using SIFT detector.

`#include <PointsToTrackWithImage.h>`

Inheritance diagram for OpencvSfM::PointsToTrackWithImage:



Public Member Functions

- PointsToTrackWithImage (cv::Mat imageToAnalyse, cv::Mat maskOfAnalyse, cv::Ptr< cv::FeatureDetector > feature_detector=0, cv::Ptr< cv::DescriptorExtractor > descriptor_detector=0)
- PointsToTrackWithImage (cv::Mat imageToAnalyse, cv::Mat maskOfAnalyse, std::string feature_detector, std::string descriptor_detector="SIFT")
- void setFeatureDetector (cv::Ptr< cv::FeatureDetector > feature_detector)
- void setDescriptorExtractor (cv::Ptr< cv::DescriptorExtractor > descriptor_detector)
- int computeKeypoints ()
- void computeDescriptors ()

Protected Attributes

- cv::Ptr< cv::FeatureDetector > feature_detector_

  class which will find the points
- cv::Ptr< cv::DescriptorExtractor > descriptor_detector_

  class which will compute the descriptors
- cv::Mat imageToAnalyse_

  Picture from where points are detected.
- cv::Mat maskOfAnalyse_

  Mask of analyse. Everything out of this mask is ignored.

3.8.1 Detailed Description

This class can be used to find points and features in pictures using SIFT detector.

To create a structure from motion, most methods use points to compute the structure. This class focus on the first task in SfM: find points in image which are easy to track... When available, a feature vector for each points is very helpful: the matching will be easier.

Definition at line 14 of file PointsToTrackWithImage.h.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 OpencvSfM::PointsToTrackWithImage::PointsToTrackWithImage (cv::Mat *imageToAnalyse*, cv::Mat *maskOfAnalyse*, cv::Ptr< cv::FeatureDetector > *feature_detector* = 0, cv::Ptr< cv::DescriptorExtractor > *descriptor_detector* = 0)

First constructor used to create a list of points to track using a feature and a descriptor algorithm.

Parameters

imageToAnalyse	Image to use for keypoints and features search
maskOfAnalyse	Mask used to hide part of image
feature_detector	Algorithm to use for features detection (see http://opencv.willowgarage.com/documentation/cpp/common_interfaces_for_feature_detection_and_descriptor_extraction.html#featuredetector)
descriptor_detector	Algorithm to use for descriptors detection (see http://opencv.willowgarage.com/documentation/cpp/common_interfaces_for_feature_detection_and_descriptor_extraction.html#descriptorextractor)

3.8.2.2 OpencvSfM::PointsToTrackWithImage::PointsToTrackWithImage (cv::Mat *imageToAnalyse*, cv::Mat *maskOfAnalyse*, std::string *feature_detector*, std::string *descriptor_detector* = "SIFT")

Second constructor used to create a list of points to track using a feature and a descriptor algorithm.

Parameters

imageToAnalyse	Image to use for keypoints and features search
maskOfAnalyse	Mask used to hide part of image
feature_detector	name of the algorithm to use for features detection (see http://opencv.willowgarage.com/documentation/cpp/common_interfaces_for_feature_detection_and_descriptor_extraction.html#featuredetector)
descriptor_detector	name of the algorithm to use for descriptors detection (see http://opencv.willowgarage.com/documentation/cpp/common_interfaces_for_feature_detection_and_descriptor_extraction.html#descriptorextractor)

3.8.3 Member Function Documentation

3.8.3.1 void OpencvSfM::PointsToTrackWithImage::computeDescriptors ( ) [virtual]

This method is used to compute only descriptors...

Reimplemented from OpencvSfM::PointsToTrack.

Definition at line 45 of file PointsToTrackWithImage.cpp.

3.8.3.2 int OpencvSfM::PointsToTrackWithImage::computeKeypoints ( ) [virtual]

This method is used to compute only Keypoints...

Returns

the number of points

Reimplemented from OpencvSfM::PointsToTrack.

Definition at line 39 of file PointsToTrackWithImage.cpp.

3.8.3.3 void OpencvSfM::PointsToTrackWithImage::setDescriptorExtractor (cv::Ptr< cv::DescriptorExtractor > *descriptor_detector*)

Use this function to set the descriptor extractor. Can be useful to update parameters, for example!

Parameters

descriptor_detector	new pointer of a descriptor extractor algorithm.

Definition at line 28 of file PointsToTrackWithImage.cpp.

3.8.3.4 void OpencvSfM::PointsToTrackWithImage::setFeatureDetector (cv::Ptr< cv::FeatureDetector > *feature_detector*)

Use this function to set the feature detector. Can be useful to update parameters, for example!

Parameters

feature_detector	new pointer of a feature detector algorithm.

Definition at line 23 of file PointsToTrackWithImage.cpp.

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

- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/PointsToTrackWithImage.h
- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/PointsTo-

TrackWithImage.cpp

Index