GSoC2011SfM

0.1

Generated by Doxygen 1.7.4

Mon May 30 2011 17:50:29

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

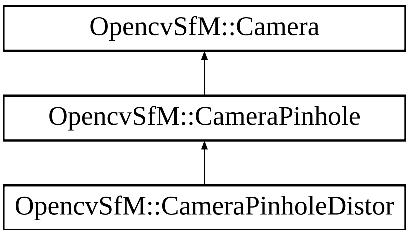
Class Documentation

3.1 OpencvSfM::Camera Class Reference

We will need this class, but PointOfView need our class too...

`#include <Camera.h>`

Inheritance diagram for OpencvSfM::Camera:



Public Member Functions

- virtual std::vector$<$ cv::Vec4d $>$ convertFromImageTo3Dray (std::vector$<$ cv::Vec3d $>$ points)=0
- virtual std::vector$<$ cv::Vec2d $>$ pixelToNormImageCoordinates (std::vector$<$ cv::Vec2d $>$ points)=0
- virtual std::vector$<$ cv::Vec2d $>$ normImageToPixelCoordinates (std::vector$<$ cv::Vec2d $>$ points)=0
- virtual cv::Mat computeProjectionMatrix (const cv::Mat &rotation, const cv::Vec3d &translation)

Protected Attributes

- std::vector$<$ PointOfView $*$ $>$ pointsOfView_

  vector of the differents positions of the camera.

Friends

- class **PointOfView**

3.1.1 Detailed Description

We will need this class, but PointOfView need our class too...

This class represent the physical device which take the pictures. It is not related to a 3D position which is the role of the PointOfView class. The role of the class is to store only device related informations like intra parameters, radial and tangential distotion. This abstract class is not related to a type of camera (fish eyes...)

This class can be used to store device related informations like intra parameters, radial and tangential distortion. If we use the so-called pinhole camera model, 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:

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

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

$$\text{where} \quad r^2 = x'^2 + y'^2$$
$$u = f_x * x'' + c_x$$
$$v = f_y * y'' + c_y$$

radial_dist_ can be used to store k_1 to k_6 tangential_dist_ can be used to store p_1 and p_2

So this class is devoted to the conversion between 2D points from pixel image coordinates and 2D points in normalized image coordinates, or ray projection using intra parameters.

Definition at line 47 of file Camera.h.

3.1.2 Member Function Documentation

3.1.2.1 virtual cv::Mat OpencvSfM::Camera::computeProjectionMatrix (const cv::Mat & *rotation*, const cv::Vec3d & *translation*) [virtual]

This method can create a projection matrix using intra parameters and given rotation and translation As we don't have intra parameters, this method only compute matrix [R|t]

Parameters

rotation	rotation matrix
translation	translation vector

Returns

Projection matrix (4∗3)

Reimplemented in OpencvSfM::CameraPinhole.

3.1.2.2 virtual std::vector<cv::Vec4d> OpencvSfM::Camera::convertFromImageTo3Dray (std::vector< cv::Vec3d > *points*) [pure virtual]

This method can transform points from image to 3D rays (homogeneous coordinates)

Implemented in OpencvSfM::CameraPinhole, and OpencvSfM::CameraPinholeDistor.

3.1.2.3 virtual std::vector<cv::Vec2d> OpencvSfM::Camera::normImageToPixelCoordinates (std::vector< cv::Vec2d > *points*) [pure virtual]

This method can convert 2D points from normalized image coordinates to 2D points in pixel image coordinates

Parameters

points	2D points in normalized image homogeneous coordinates.

Returns

2D points in pixel image homogeneous coordinates.

Implemented in OpencvSfM::CameraPinhole, and OpencvSfM::CameraPinholeDistor.

3.1.2.4 virtual std::vector<cv::Vec2d> OpencvSfM::Camera::pixelToNormImageCoordinates (std::vector< cv::Vec2d > *points*) [pure virtual]

This method can convert 2D points from pixel image coordinates to 2D points in normalized image coordinates

Parameters

points	2D points in pixel image homogeneous coordinates.

Returns

2D points in normalized image homogeneous coordinates.

Implemented in OpencvSfM::CameraPinhole, and OpencvSfM::CameraPinholeDistor.

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

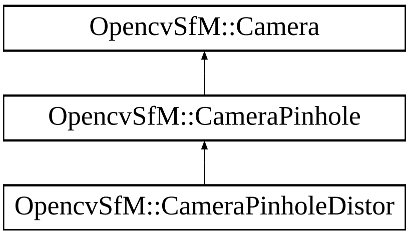
- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/Camera.h
- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/Camera.cpp

3.2 OpencvSfM::CameraPinhole 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 PointOfView class. The role of the class is to store only intra parameters (without radial distortion)

```
#include <CameraPinhole.h>
```

Inheritance diagram for OpencvSfM::CameraPinhole:



Public Member Functions

- CameraPinhole (cv::Mat intra_params=cv::Mat::eye(3, 3, CV_64F), unsigned char wantedEstimation=FOCAL_PARAM|SKEW_PARAM|PRINCIPAL_POINT_PARAM)
- CameraPinhole (const std::vector< std::vector< cv::Point3f > > &objectPoints, const std::vector< std::vector< cv::Point2f > > &imagePoints, cv::Size imageSize, double aspectRatio=1., unsigned char wantedEstimation=FOCAL_PARAM|SKEW_PARAM|PRINCIPAL_POINT_PARAM)
- void updateIntrinsicMatrix (cv::Mat newParams, unsigned char intraValues=FOCAL_PARAM|SKEW_PARAM|PRINCIPAL_POINT_PARAM)
- virtual std::vector< cv::Vec4d > convertFromImageTo3Dray (std::vector< cv::Vec3d > points)
- virtual std::vector< cv::Vec2d > pixelToNormImageCoordinates (std::vector< cv::Vec2d > points)
- virtual std::vector< cv::Vec2d > normImageToPixelCoordinates (std::vector< cv::Vec2d > points)

- virtual cv::Mat computeProjectionMatrix (const cv::Mat &rotation, const cv::Vec3d &translation)

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...
- unsigned char estimation_needed_

3.2.1 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 PointOfView class. The role of the class is to store only intra parameters (without radial distortion)

So this class is devoted to the conversion between 3D points (using camera coordinate) and 2D points (using image coordinate) using the methods convertFromImageTo3Dray or convertFrom3DToImage

Definition at line 23 of file CameraPinhole.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 OpencvSfM::CameraPinhole::CameraPinhole (cv::Mat *intra_params =* `cv::Mat::eye(3, 3, CV_64F)`, unsigned char *wantedEstimation =* `FOCAL_PARAM|SKEW_PARAM|PRINCIPAL_POINT_PARAM`)

Constructor with (or not) intra parameters.

Parameters

intra_params	matrix of intra parameters (3∗3)
wantedEstimation	values which need an estimation

3.2.2.2 OpencvSfM::CameraPinhole::CameraPinhole (const std::vector< std::vector< cv::Point3f > > & *objectPoints,* const std::vector< std::vector< cv::Point2f > > & *imagePoints,* cv::Size *imageSize,* double *aspectRatio =* `1.`, unsigned char *wantedEstimation =* `FOCAL_PARAM|SKEW_PARAM|PRINCIPAL_POINT_PARAM`)

Constructor where initial camera matrix is computed from the 3D-2D point correspondences. Currently, the function only supports planar calibration patterns, i.e. patterns where each object point has z-coordinate =0.

Parameters

objectPoints	The vector of vectors of the object points. See http://opencv.willowgarage.com/documentation/cpp/calib3d_-camera_calibration_and_3d_reconstruction.html#cv-calibratecamera
imagePoints	The vector of vectors of the corresponding image points. See http://opencv.willowgarage.com/documentation/cpp/calib3d_-camera_calibration_and_3d_reconstruction.html#cv-calibratecamera
imageSize	The image size in pixels; used to initialize the principal point
aspectRatio	If it is zero or negative, both f_x and f_y are estimated independently. Otherwise $f_x = f_y * aspectRatio$
wantedEstimation	values which need an estimation

3.2.3 Member Function Documentation

3.2.3.1 Mat OpencvSfM::CameraPinhole::computeProjectionMatrix (const cv::Mat & *rotation*, const cv::Vec3d & *translation*) [virtual]

This method can create a projection matrix using intra parameters and given rotation and translation

Parameters

rotation	rotation matrix
translation	translation vector

Returns

Projection matrix (4∗3)

Reimplemented from OpencvSfM::Camera.

Definition at line 19 of file Camera.cpp.

3.2.3.2 virtual std::vector<cv::Vec4d> OpencvSfM::CameraPinhole::convertFromImageTo3Dray (std::vector< cv::Vec3d > *points*) [virtual]

This method can transform points from image to 3D rays

Implements OpencvSfM::Camera.

Reimplemented in OpencvSfM::CameraPinholeDistor.

3.2.3.3 vector< Vec2d > OpencvSfM::CameraPinhole::normImageToPixelCoordinates (std::vector< cv::Vec2d > *points*) [virtual]

This method can convert 2D points from normalized image coordinates to 2D points in pixel image coordinates

Parameters

points	2D points in normalized image homogeneous coordinates.

Returns

2D points in pixel image homogeneous coordinates.

Implements OpencvSfM::Camera.

Reimplemented in OpencvSfM::CameraPinholeDistor.

Definition at line 100 of file CameraPinhole.cpp.

3.2.3.4 virtual std::vector<cv::Vec2d> OpencvSfM::CameraPinhole::pixelToNormImageCoordinates (std::vector< cv::Vec2d > *points*) [virtual]

This method can convert 2D points from pixel image coordinates to 2D points in normalized image coordinates

Parameters

points	2D points in pixel image homogeneous coordinates.

Returns

2D points in normalized image homogeneous coordinates.

Implements OpencvSfM::Camera.

Reimplemented in OpencvSfM::CameraPinholeDistor.

3.2.3.5 void OpencvSfM::CameraPinhole::updateIntrinsicMatrix (cv::Mat *newParams*, unsigned char *intraValues* = FOCAL_PARAM|SKEW_PARAM|PRINCIPAL_POINT_PARAM)

this method can be used to update the intra parameters.

Parameters

newParams	matrix of new parameters (3∗3)
intraValues	values which are useful in matrix

Definition at line 34 of file CameraPinhole.cpp.

3.2.4 Member Data Documentation

3.2.4.1 unsigned char OpencvSfM::CameraPinhole::estimation_needed_ [protected]

This attribut is used to know what we should estimate... Example: if equal to 0, nothing should be estimated... If equal to 3, focal and skew should be estimated (FOCAL_PARAM + SKEW_PARAM)

Definition at line 33 of file CameraPinhole.h.

Referenced by OpencvSfM::CameraPinholeDistor::updateDistortionParameters().

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

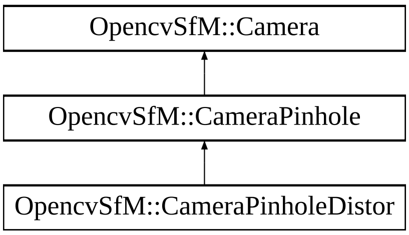
- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/CameraPinhole.h
- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/Camera.cpp
- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/CameraPinhole.cpp

3.3 OpencvSfM::CameraPinholeDistor 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 PointOfView class. The role of the class is to store intra parameters and radial distortion.

`#include <CameraPinholeDistor.h>`

Inheritance diagram for OpencvSfM::CameraPinholeDistor:



Public Member Functions

- CameraPinholeDistor (cv::Mat intra_params=cv::Mat::eye(3, 3, CV_64F), cv::Vec6d radial_dist=cv::Vec6d(0.0, 0.0, 0.0, 0.0, 0.0, 0.0), unsigned char nbRadialParam=6, cv::Vec2d tangential_dist=cv::Vec2d(0.0, 0.0), unsigned char wantedEstimation=FOCAL_PARAM|SKEW_PARAM|PRINCIPAL_POINT_PARAM|RADIAL_PARAM|TANGEANT_PARAM)
- CameraPinholeDistor (const std::vector< std::vector< cv::Point3f > > &objectPoints, const std::vector< std::vector< cv::Point2f > > &imagePoints, cv::Size imageSize, double aspectRatio=1., cv::Vec6d radial_dist=cv::Vec6d(0.0, 0.0, 0.0, 0.0, 0.0, 0.0), unsigned char nbRadialParam=6, cv::Vec2d tangential_dist=cv::Vec2d(0.0,

0.0), unsigned char wantedEstimation=FOCAL_PARAM|SKEW_PARAM|PRINCIPAL_POINT_PARAM|RADIAL_PARAM|TANGEANT_PARAM)

- void updateDistortionParameters (const cv::Vec6d &radial_dist, unsigned char nbRadialParam, const cv::Vec2d &tangential_dist, unsigned char wantedEstimation=RADIAL_PARAM|TANGEANT_PARAM)
- virtual std::vector< cv::Vec4d > convertFromImageTo3Dray (std::vector< cv::Vec3d > points)
- virtual std::vector< cv::Vec2d > pixelToNormImageCoordinates (std::vector< cv::Vec2d > points)
- virtual std::vector< cv::Vec2d > normImageToPixelCoordinates (std::vector< cv::Vec2d > points)

Protected Attributes

- cv::Vec< double, 6 > radial_dist_

  used to store radial dist parameters (/fk_1/f to /fk_6/f)
- unsigned char nb_radial_params_

  number of radial dist parameters (0, 2, 3 or 6)
- cv::Vec< double, 2 > tangential_dist_

  used to store tangential dist parameters (/fp_1/f and /fp_2/f)
- unsigned char **nb_tangent_params_**
- cv::Mat distortionVector

  vector of distortion coefficients (k_1, k_2, p_1, p_2[, k_3[, k_4, k_5, k_6]]) of 4, 5 or 8 elements
- cv::Mat undistMapX

  Undistortion and rectification transformation map.
- cv::Mat undistMapY

  Undistortion and rectification transformation map.

3.3.1 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 PointOfView class. The role of the class is to store intra parameters and radial distortion.

So this class is devoted to the conversion between 3D points (using camera coordinate) and 2D points (using image coordinate) using the methods convertFromImageTo3Dray or convertFrom3DToImage

Definition at line 23 of file CameraPinholeDistor.h.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 OpencvSfM::CameraPinholeDistor::CameraPinholeDistor (cv::Mat *intra_params* = cv::Mat::eye(3, 3, CV_64F), cv::Vec6d *radial_dist* = cv::Vec6d(0.0, 0.0, 0.0, 0.0, 0.0, 0.0), unsigned char *nbRadialParam* = 6, cv::Vec2d *tangential_dist* = cv::Vec2d(0.0, 0.0), unsigned char *wantedEstimation* = FOCAL_PARAM|SKEW_PARAM|PRINCIPAL_POINT_PARAM|RADIAL_PARAM|TANGEANT_PARAM)

Constructor with (or not) intra parameters.

Parameters

intra_params	matrix of intra parameters (3$*$3)
radial_dist	radial dist parameters (/fk_1/f to /fk_6/f)
nbRadialParam	number of radial dist parameters (0, 2, 3 or 6)
tangential_dist	tangential dist parameters (/fp_1/f and /fp_2/f)
wantedEstimation	values which need an estimation

3.3.2.2 OpencvSfM::CameraPinholeDistor::CameraPinholeDistor (const std::vector< std::vector< cv::Point3f > > & *objectPoints*, const std::vector< std::vector< cv::Point2f > > & *imagePoints*, cv::Size *imageSize*, double *aspectRatio* = 1., cv::Vec6d *radial_dist* = cv::Vec6d(0.0, 0.0, 0.0, 0.0, 0.0, 0.0), unsigned char *nbRadialParam* = 6, cv::Vec2d *tangential_dist* = cv::Vec2d(0.0, 0.0), unsigned char *wantedEstimation* = FOCAL_PARAM|SKEW_PARAM|PRINCIPAL_POINT_PARAM|RADIAL_PARAM|TANGEANT_PARAM)

Constructor where initial camera matrix is computed from the 3D-2D point correspondences. Currently, the function only supports planar calibration patterns, i.e. patterns where each object point has z-coordinate =0.

Parameters

objectPoints	The vector of vectors of the object points. See http://opencv.willowgarage.com/documentation/cpp/calib3d_camera_calibration_and_3d_reconstruction.html#cv-calibratecamera
imagePoints	The vector of vectors of the corresponding image points. See http://opencv.willowgarage.com/documentation/cpp/calib3d_camera_calibration_and_3d_reconstruction.html#cv-calibratecamera
imageSize	The image size in pixels; used to initialize the principal point
aspectRatio	If it is zero or negative, both f_x and f_y are estimated independently. Otherwise $f_x = f_y * aspectRatio$
radial_dist	radial dist parameters (/fk_1/f to /fk_6/f)
nbRadialParam	number of radial dist parameters (0, 2, 3 or 6)

tangential_dist	tangential dist parameters (/fp_1/f and /fp_2/f)
wantedEstimation	values which need an estimation

3.3.3 Member Function Documentation

3.3.3.1 vector< Vec4d > OpencvSfM::CameraPinholeDistor::convertFromImageTo3Dray (std::vector< cv::Vec3d > *points*) `[virtual]`

This method can transform points from image to 3D rays

Reimplemented from OpencvSfM::CameraPinhole.

Definition at line 65 of file CameraPinhole.cpp.

3.3.3.2 vector< Vec2d > OpencvSfM::CameraPinholeDistor::normImageToPixelCoordinates (std::vector< cv::Vec2d > *points*) `[virtual]`

This method can convert 2D points from normalized image coordinates to 2D points in pixel image coordinates

Parameters

points	2D points in normalized image homogeneous coordinates.

Returns

2D points in pixel image homogeneous coordinates.

Reimplemented from OpencvSfM::CameraPinhole.

Definition at line 74 of file CameraPinholeDistor.cpp.

3.3.3.3 vector< Vec2d > OpencvSfM::CameraPinholeDistor::pixelToNormImageCoordinates (std::vector< cv::Vec2d > *points*) `[virtual]`

This method can convert 2D points from pixel image coordinates to 2D points in normalized image coordinates

Parameters

points	2D points in pixel image homogeneous coordinates.

Returns

2D points in normalized image homogeneous coordinates.

Reimplemented from OpencvSfM::CameraPinhole.

Definition at line 71 of file CameraPinhole.cpp.

3.3.3.4 void OpencvSfM::CameraPinholeDistor::updateDistortionParameters (const cv::Vec6d & *radial_dist,* unsigned char *nbRadialParam,* const cv::Vec2d & *tangential_dist,* unsigned char *wantedEstimation =* RADIAL_PARAM|TANGEANT_PARAM)

this method can be used to update the intra parameters.

Parameters

radial_dist	values of the new radial distortions parameters
nbRadialParam	number of radial dist parameters (0, 2, 3 or 6)
tangential_dist	values of the new tangential distortions parameters
wantedEstimation	values which need an estimation

Definition at line 29 of file CameraPinholeDistor.cpp.

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

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

3.4 OpencvSfM::MotionEstimator Class Reference

3.4.1 Detailed Description

Definition at line 5 of file MotionEstimator.h.

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

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

3.5 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.

3.5.1 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.

3.5.2 Member Function Documentation

3.5.2.1 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.

3.5.2.2 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.

3.5.2.3 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.

3.5.2.4 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

3.5.2.5 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

3.5.2.6 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.

3.5.3 Member Data Documentation

3.5.3.1 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.

3.5.3.2 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

3.6 OpencvSfM::PointOfView 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 <PointOfView.h>`

Public Member Functions

- PointOfView (cv::Ptr< Camera > device, cv::Mat rotation=cv::Mat::eye(3, 3, CV_64F), cv::Vec3d translation=cv::Vec3d(0.0, 0.0, 0.0))
- virtual ~PointOfView (void)
- cv::Ptr< Camera > getIntraParameters () const

Protected Attributes

- cv::Mat rotation_

  Rotation matrix R.
- cv::Vec3d translation_

  Translation vector t.
- cv::Mat projection_matrix_

  redundancy but speed improvement

- cv::Ptr< Camera > device_

  intra parameters and distortion coefs
- unsigned char config_

  This attribut is used to know what we should estimate... If equal to 0, nothing should be estimated...

3.6.1 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 42 of file PointOfView.h.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 OpencvSfM::PointOfView::PointOfView (cv::Ptr< Camera > *device*, cv::Mat *rotation* = `cv::Mat::eye(3, 3, CV_64F)`, cv::Vec3d *translation* = `cv::Vec3d(0.0,0.0,0.0)`)

To create a point of view, we need two things : a camera, and a point (with orientation). 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...
rotation	Matrix of the known rotation (optional)...
translation	Vector of the known translation (optional)...

<Rotation matrix R

<Translation vector t

Definition at line 14 of file PointOfView.cpp.

3.6.2.2 OpencvSfM::PointOfView::∼PointOfView (void) [virtual]

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

Definition at line 31 of file PointOfView.cpp.

3.6.3 Member Function Documentation

3.6.3.1 cv::Ptr<Camera> OpencvSfM::PointOfView::getIntraParameters ( ) const [inline]

use this function to get acces to the camera parameters

Returns

camera matrix

Definition at line 71 of file PointOfView.h.

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

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

3.7 OpencvSfM::Points3DEstim Class Reference

3.7.1 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

3.8 OpencvSfM::PointsMatcher Class Reference

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

```
#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_**

3.8.1 Detailed Description

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

Definition at line 12 of file PointsMatcher.h.

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

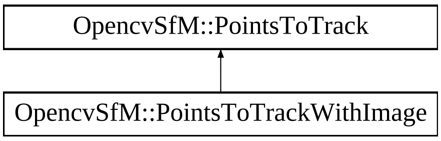
- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/PointsMatcher.h
- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/PointsMatcher.cpp

3.9 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 PointsToTrackWithImage.

```
#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.

3.9.1 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 PointsToTrackWithImage.

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.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 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.

3.9.2.2 OpencvSfM::PointsToTrack::∼PointsToTrack (void) [virtual]

Destructor : delete points and features vectors

Definition at line 17 of file PointsToTrack.cpp.

3.9.3 Member Function Documentation

3.9.3.1 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
computeMissingDescriptor	if true, the missing descriptors are computed.

Definition at line 41 of file PointsToTrack.cpp.

3.9.3.2 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().

3.9.3.3 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().

3.9.3.4 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.

3.9.3.5 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.

3.9.3.6 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.

3.9.3.7 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:

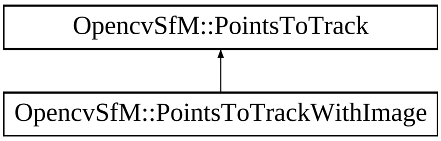
- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/PointsToTrack.h
- D:/Travail/These/Determination caracteristiques camera/GSoC/SfM/src/PointsToTrack.cpp

3.10 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.

3.10.1 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.

3.10.2 Constructor & Destructor Documentation

3.10.2.1 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`)

3.10.2.2 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)

3.10.3 Member Function Documentation

3.10.3.1 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.

3.10.3.2 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.

3.10.3.3 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.

3.10.3.4 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/PointsToTrackWithImage.cpp

Index