Kudan SLAM: User Manual

Generated by Doxygen 1.8.11

Contents

Chapter 1

Kudan SLAM

This is the **user manual** for the Kudan SLAM system. This provides documentation for the public interface to the SLAM system, accessible using the *KudanSLAM* class.

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 KudanFrame Class Reference

A representation of a camera frame, for input to the SLAM system.

`#include <Interface.hpp>`

Public Attributes

- KudanImage leftImage

  Left greyscale image (for stereo, or the single monocular image)
- KudanImage rightImage

  Right greyscale image for a stereo pair.

3.1.1 Detailed Description

A representation of a camera frame, for input to the SLAM system.

This will contain image data (for one or more cameras), plus other inputs (e.g. IMU)

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

- SLAM/Interface.hpp

3.2 KudanImage Class Reference

A basic representation of an image.

`#include <Interface.hpp>`

Public Member Functions

- KudanImage ()

  Default initialisation (ensures that data are null and size is zero unless specified)

Public Attributes

- size_t width

  Width of the image. Defaults to 0 if not initialised.
- size_t height

  Height of the image. Defaults to 0 if not initialised.
- unsigned char * data

  *The image data, for a single channel 8-bit image. The size allocated to this must correspond to width * height. Defaults to NULL if not initialised.*

3.2.1 Detailed Description

A basic representation of an image.

The image data are represented as 8-bit (unsigned char) arrays. This assumes rectangular single-channel (greyscale) images.

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

- SLAM/Interface.hpp

3.3 KudanMatrix3 Class Reference

A simple representation of a 3x3 matrix of floating point numbers.

```
#include <Interface.hpp>
```

Public Attributes

- float data [9]

  Representation of matrix data in row-major format (i.e. first row, then second row, etc)

3.3.1 Detailed Description

A simple representation of a 3x3 matrix of floating point numbers.

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

- SLAM/Interface.hpp

3.4 KudanMatrix4 Class Reference

A simple representation of a 4x4 matrix of floating point numbers.

`#include <Interface.hpp>`

Public Attributes

- float data [16]
  Representation of matrix data in row-major format (i.e. first row, then second row, etc)

3.4.1 Detailed Description

A simple representation of a 4x4 matrix of floating point numbers.

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

- SLAM/Interface.hpp

3.5 KudanQuaternion Class Reference

A simple representation of a 3D orientation, as a quaternion.

`#include <Interface.hpp>`

Public Attributes

- float x
  X coordinate.
- float y
  Y coordinate.
- float z
  Z coordinate.
- float w
  W coordinate.

3.5.1 Detailed Description

A simple representation of a 3D orientation, as a quaternion.

This provides a convenient class for representing the data, but no methods for manipulation.

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

- SLAM/Interface.hpp

3.6 KudanSLAM Class Reference

This class represents a whole SLAM system.

`#include <Interface.hpp>`

Public Types

- enum State {
  State::Idle, State::Initialising, State::Tracking, State::Lost,
  State::Found }

  This describes the states that the SLAM system can be in, from initialisation through to tracking, including failure and recovery.
- enum ProcessFrameError {
  ProcessFrameError::EverythingIsFine, ProcessFrameError::InputImageError, ProcessFrameError::↩
  CalibrationError, ProcessFrameError::InitialisationError,
  ProcessFrameError::Unknown }

  This describes different return conditions for the processFrame function, describing what (if anything) went wrong.
- enum CameraParameterError {
  CameraParameterError::EverythingIsFine, CameraParameterError::BadSize, CameraParameterError::Bad↩
  FocalLength, CameraParameterError::BadPrincipalPoint,
  CameraParameterError::InvalidCalibration, CameraParameterError::Unknown }

  This describes things that can go wrong when trying to set camera parameters.

Public Member Functions

- KudanSLAM ()

  Default constructor for the SLAM system interface.
- void startTracking ()

  This sets the initialisation variable, so SLAM will initialise on the next frame.
- ProcessFrameError processFrame (KudanFrame &frame)

  This is the function with which new data are passed into SLAM to process.
- CameraParameterError setCameraCalibration (float imageWidth, float imageHeight, float focalLengthX, float focalLengthY, float principalPointX, float principalPointY)

  Set up the camera intrinsic calibration.
- CameraParameterError setCameraDistortion (std::vector< float > distortionCoefficients)

  Set the radial and tangential distortion parameters for the camera.
- CameraParameterError setCameraDistortion (std::vector< float > distortionCoefficientsLeft, std::vector< float > distortionCoefficientsRight)

  Set the radial and tangential distortion parameters for the left and right cameras separately for a stereo pair.
- CameraParameterError setCameraRectification (KudanMatrix3 intrinsicsLeft, KudanMatrix3 rotationLeft, std::vector< float > distortionCoefficientsLeft, KudanMatrix3 intrinsicsRight, KudanMatrix3 rotationRight, std::vector< float > distortionCoefficientsRight)

  Set the rectification extrinsics for a stereo camera pair, plus undistortion for each individual camera.
- CameraParameterError setCameraRectification (float fx_l, float fy_l, float cx_l, float cy_l, KudanMatrix3 rotationLeft, std::vector< float > distortionCoefficientsLeft, float fx_r, float fy_r, float cx_r, float cy_r, Kudan↩
  Matrix3 rotationRight, std::vector< float > distortionCoefficientsRight)

  Set the rectification extrinsics for a stereo camera pair, plus undistortion for each individual camera.
- void resetCameraDistortion ()

  If distortion parameters have been set, this clears them.
- bool frameWasUndistorted ()

Query whether the last frame processed by SLAM was automatically undistorted or not.

- KudanVector2 getFocalLength ()

  Accessor for the camera's focal length.
- KudanVector2 getPrincipalPoint ()

  Accessor for the camera's principal point.
- KudanMatrix3 getCalibrationMatrix ()

  Accessor for the camera calibration intrinsics matrix.
- void reset ()

  Reset the SLAM system.
- KudanSLAM::State getState ()

  Get the current SLAM state (e.g.
- bool saveSnapshot ()

  Save the current SLAM status of all requested variables (camera, 2D/3D points etc).
- void setRequestCameraPoses (bool request)

  Request that the camera poses (represented as rotation matrices, quaternions and translations in camera and world space) should be stored on the next run.
- void setRequestCameraMatrices (bool request)

  Request that the camera poses (represented as 4x4 matrices in camera and world space) should be stored on the next run.
- void setRequestPoints3D (bool request)

  Set whether 3D points are to be saved during the snapshot.
- void setRequestPoints2D (bool request)

  Set whether 2D points are to be saved during the snapshot.
- bool setMonocularMode ()

  Set that the SLAM system will be running in monocular mode.
- bool setStereoMode ()

  Set that the SLAM system will be running in stereo mode.
- bool setPatchMode ()

  Set the SLAM system to use patches for tracking and expansion.
- bool setDescriptorMode ()

  Set the SLAM system to use descriptors for tracking and expansion.
- void setShouldExpand (bool shouldExpand)

  Request expansion now.
- std::vector< KudanVector2 > getMapPoints2D ()

  Get the map points as projected into 2D by the current camera pose (only those which are in view)
- std::vector< KudanVector2 > getTrackedPoints2D ()

  Get the current tracked points in the 2D image.
- std::vector< KudanVector3 > getMapPoints3D ()

  Get the full set of map points in the 3D world coordinate frame.
- std::vector< KudanVector3 > getVisiblePoints3D ()

  Get the subset of 3D points which are currently visible from the current camera (corresponds to MapPoints2D)
- std::vector< KudanVector3 > getTrackedPoints3D ()

  Get the subset of 3D points (in the world coordinate frame) which were tracked in the last frame.
- KudanVector3 getCameraTranslation ()

  Get the camera translation.
- KudanMatrix3 getCameraRotation ()

  Get the camera rotation.
- KudanQuaternion getCameraQuaternion ()

  Get the camera rotation as a quaternion.
- KudanVector3 getCameraPosition ()

  Get the position of the camera centre in the world coordinate frame (camera transform form)

- KudanMatrix3 getCameraOrientationInWorld ()

  Get the camera orientation in the world coordinate frame, about its own centre (camera transform form)
- KudanQuaternion getCameraQuaternionInWorld ()

  Get the camera orientation in the world coordinate frame, as a quaternion, about its own centre (camera transform form)
- KudanMatrix4 getCameraTransformMatrix ()

  Get a 4x4 matrix representing the camera transformation.
- KudanMatrix4 getCameraViewMatrix ()

  Get a 4x4 matrix representing the camera view matrix.
- int getNumKeyframes ()

  Get the total number of keyframes accessible to the tracker.
- int getNumGoodKeyframes ()

  Get the number of keyframes accessible to the tracker which are not culled.
- float getTrackingTime ()

  Get the time taken for the last frame to be processed The processing time for the last frame in seconds.
- float getTrackingRate ()

  Get the effective frame rate as of the past processed frame.
- bool isBundleAdjusting ()

  Is bundle adjustment happening right now?

Static Public Member Functions

- static std::string describeSlamState (KudanSLAM::State state)

  The SLAM state is represented by an enum: this function converts to a readable string if possible.
- static std::string describeErrorCode (ProcessFrameError code)

  One of many overloaded error-code stringifying functions.
- static std::string describeErrorCode (CameraParameterError code)

  One of many overloaded error-code stringifying functions.
- static std::string version ()

  Print SLAM version number if available.

3.6.1 Detailed Description

This class represents a whole SLAM system.

Create an object of this to run SLAM!

3.6.2 Member Enumeration Documentation

3.6.2.1 enum KudanSLAM::CameraParameterError [strong]

This describes things that can go wrong when trying to set camera parameters.

Enumerator

EverythingIsFine No error, everything was fine.

BadSize The size is invalid, might be zero or negative.

BadFocalLength The focal length is invalid, might be zero or negative.

BadPrincipalPoint The focal length is invalid, might be zero or negative, or more than the image size (principal point must lie within the image)

InvalidCalibration The calibration used for this operation was not valid.

Unknown Unknown error occurred / error type is undefined.

3.6.2.2 enum KudanSLAM::ProcessFrameError [strong]

This describes different return conditions for the processFrame function, describing what (if anything) went wrong.

Enumerator

EverythingIsFine No error: processing was fine (tracking might not be!)

InputImageError A problem with the input image. Check the input images are provided and the data are the right size / not null.

CalibrationError A problem with the camera calibration. Check it has been given to SLAM and is valid.

InitialisationError The frame was processed for initialisation but this failed for some reason. NOTE: This is not an error as such, as initialisation can fail (depending on input) but might be indicative of a deeper problem.

Unknown Unknown error occurred / error type is undefined.

3.6.2.3 enum KudanSLAM::State [strong]

This describes the states that the SLAM system can be in, from initialisation through to tracking, including failure and recovery.

Enumerator

Idle Not doing anything, especially before initialisation.

Initialising Attempting to initialise tracking.

Tracking Tracking is happening.

Lost Tracking has been lost (recovery might be in progress)

Found Tracking was recently recovered (intermediate careful state)

3.6.3 Constructor & Destructor Documentation

3.6.3.1 KudanSLAM::KudanSLAM ( )

Default constructor for the SLAM system interface.

This will initialise everything to default settings.

3.6.4 Member Function Documentation

3.6.4.1 static std::string KudanSLAM::describeSlamState (KudanSLAM::State *state*) [static]

The SLAM state is represented by an enum: this function converts to a readable string if possible.

Note this is a static function, it takes a state as input, it does not return the current SLAM state

Parameters

state	The state to be described

Returns

string representation of the input state

3.6.4.2 KudanMatrix3 KudanSLAM::getCalibrationMatrix ( )

Accessor for the camera calibration intrinsics matrix.

This is a 3x3 matrix encoding the focal length and principal point (skew is always zero), suitable for use in projection from 3D to 2D.

Returns

A KudanMatrix3 class representing the 3x3 intrinsics matrix

3.6.4.3 KudanMatrix3 KudanSLAM::getCameraOrientationInWorld ( )

Get the camera orientation in the world coordinate frame, about its own centre (camera transform form)

Returns

A KudanMatrix3 representing the camera orientation (world coordinate frame)

3.6.4.4 KudanVector3 KudanSLAM::getCameraPosition ( )

Get the position of the camera centre in the world coordinate frame (camera transform form)

Returns

A KudanVector3 representing the camera position (world coordinate frame)

3.6.4.5 KudanQuaternion KudanSLAM::getCameraQuaternion ( )

Get the camera rotation as a quaternion.

This is the quaternion for of 'R' in the projection matrix P = K[R|T], i.e. the orientation of the camera about the world coordinate frame, not the orientation of the camera about its centre

Returns

A KudanQuaternion representing the camera orientation (camera coordinate frame)

3.6.4.6 KudanQuaternion KudanSLAM::getCameraQuaternionInWorld ( )

Get the camera orientation in the world coordinate frame, as a quaternion, about its own centre (camera transform form)

Returns

A KudanMatrix3 representing the camera orientation (world coordinate frame)

3.6.4.7 KudanMatrix3 KudanSLAM::getCameraRotation ( )

Get the camera rotation.

This is the 'R' in the projection matrix P = K[R|T], i.e. the orientation of the camera about the world coordinate frame, not the orientation of the camera about its centre

Returns

A KudanMatrix3 representing the camera orientation (camera coordinate frame)

3.6.4.8 KudanMatrix4 KudanSLAM::getCameraTransformMatrix ( )

Get a 4x4 matrix representing the camera transformation.

This is the transformation from world space to camera space, i.e. represents the camera position in the world coordinate frame. It has the form of a 4x4 matrix where the upper-left 3x3 is the rotation matrix of the camera about its centre, and the upper-right 3x1 is the camera's 3D position in the world coordinate frame.

Returns

A 4x4 matrix representing the camera transformation

3.6.4.9 KudanVector3 KudanSLAM::getCameraTranslation ( )

Get the camera translation.

This is the 'T' in the projection matrix P = K[R|T], i.e. the position of the world origin in the camera coordinate frame (*not* the camera centre in the world coordinate frame

Returns

A KudanVector3 representing the camera translation (camera coordinate frame)

3.6.4.10 KudanMatrix4 KudanSLAM::getCameraViewMatrix ( )

Get a 4x4 matrix representing the camera view matrix.

This is the transformation from camera space to world space, i.e. it represents the transformation of 3D points in the camera coordinate frame to points in the world. It is used as part of the 3D-to-2D projection operation, x = K.P.X (this is P, for a 4x4 camera K and a 4D homogeneous point X). It has the form of a 4x4 matrix where the upper-left 3x3 is the rotation matrix of the world about the camera centre, and the upper-right 3x1 is the world origin's 3D position in the camera coordinate frame (i.e. R and T)

Returns

A 4x4 matrix representing the camera transformation

3.6.4.11 KudanVector2 KudanSLAM::getFocalLength ( )

Accessor for the camera's focal length.

This should have been set already with setCameraCalibration

Returns

Focal length in the X,Y directions represented as a 2D point

3.6.4.12 std::vector<KudanVector2> KudanSLAM::getMapPoints2D ( )

Get the map points as projected into 2D by the current camera pose (only those which are in view)

Returns

Vector of KudanVector2 representing 2D points

3.6.4.13 std::vector<KudanVector3> KudanSLAM::getMapPoints3D ( )

Get the full set of map points in the 3D world coordinate frame.

Returns

Vector of KudanVector2 representing 3D points

3.6.4.14 int KudanSLAM::getNumGoodKeyframes ( )

Get the number of keyframes accessible to the tracker which are not culled.

Returns

Number of non-culled keyframes

3.6.4.15 int KudanSLAM::getNumKeyframes ( )

Get the total number of keyframes accessible to the tracker.

Returns

Number of keyframes

3.6.4.16 KudanVector2 KudanSLAM::getPrincipalPoint ( )

Accessor for the camera's principal point.

This should have been set already with setCameraCalibration

Returns

Principal point represented as a 2D point

3.6.4.17 KudanSLAM::State KudanSLAM::getState ( )

Get the current SLAM state (e.g.

initialising, tracking) as an enum of type SLAMState

Returns

The state represented as a State enum. Use describeSlamState to get a readable string

3.6.4.18 std::vector<KudanVector2> KudanSLAM::getTrackedPoints2D ( )

Get the current tracked points in the 2D image.

Returns

Vector of KudanVector2 representing 2D points

3.6.4.19 std::vector<KudanVector3> KudanSLAM::getTrackedPoints3D ( )

Get the subset of 3D points (in the world coordinate frame) which were tracked in the last frame.

Returns

Vector of KudanVector2 representing 3D points

3.6.4.20 float KudanSLAM::getTrackingRate ( )

Get the effective frame rate as of the past processed frame.

Returns

The effective frame rate for the last frame (Hz)

3.6.4.21 float KudanSLAM::getTrackingTime ( )

Get the time taken for the last frame to be processed The processing time for the last frame in seconds.

Returns

Processing time for the last frame (s)

3.6.4.22 std::vector<KudanVector3> KudanSLAM::getVisiblePoints3D ( )

Get the subset of 3D points which are currently visible from the current camera (corresponds to MapPoints2D)

Returns

Vector of KudanVector2 representing 3D points

3.6.4.23 ProcessFrameError KudanSLAM::processFrame (KudanFrame & *frame*)

This is the function with which new data are passed into SLAM to process.

Send in one frame of image/auxiliary data and it will be processed, yielding a new estimate of the camera pose on completion, and (ultimately) an updated map, after optimisation is finished.

Parameters

frame	A frame of data for processing. This could comprise a single image (mono SLAM), a stereo pair (stereo SLAM) or other data (not implemented yet!). Note that this is a *reference* and can be modified by any undistortion/rectification

Returns

Return a boolean indicating successful processing (tracking might still have failed, this reports whether processing happened correctly)

3.6.4.24 void KudanSLAM::reset ( )

Reset the SLAM system.

This will return the underlying SLAM system to its initial state, resetting all calibration and settings. Be careful to re-load the necessary settings if used.

3.6.4.25 void KudanSLAM::resetCameraDistortion ( )

If distortion parameters have been set, this clears them.

Future frames will not be undistorted.

3.6.4.26 bool KudanSLAM::saveSnapshot ( )

Save the current SLAM status of all requested variables (camera, 2D/3D points etc).

This is called automatically by the processFrame function to snapshot the state after the last update. It could be called to get the most recent state at other times (?)

Returns

Whether any state was saved

3.6.4.27 CameraParameterError KudanSLAM::setCameraCalibration (float *imageWidth,* float *imageHeight,* float *focalLengthX,* float *focalLengthY,* float *principalPointX,* float *principalPointY*)

Set up the camera intrinsic calibration.

This is required for every invocation of SLAM

Parameters

imageWidth	The size of the camera's image
imageHeight	The size of the camera's image
focalLengthX	The focal length of the camera in the X direction
focalLengthY	The focal length of the camera in the Y direction
principalPointX	The principal point of the camera (X coordinate)
principalPointY	The principal point of the camera (Y coordinate)

Returns

A CameraParameterError indicating whether the parameter combination is valid and was successfully set on the SLAM system, or otherwise a code identifying the error

3.6.4.28 CameraParameterError KudanSLAM::setCameraDistortion (std::vector< float > *distortionCoefficients*)

Set the radial and tangential distortion parameters for the camera.

If this is set, it will activate distortion mode, i.e. the image will be undistorted before passing to the SLAM tracker. NOTE: This must be called after setting the camera intrinsics.

Parameters

distortionCoefficients	Radial (k) and tangential (p) coefficients, in the standard OpenCV order [k0, k1, p0, p1, k2, k3, k4, k5]

Returns

A CameraParameterError indicating whether the distortion was set up correctly, and otherwise what went wrong

3.6.4.29 CameraParameterError KudanSLAM::setCameraDistortion (std::vector< float > *distortionCoefficientsLeft,* std::vector< float > *distortionCoefficientsRight*)

Set the radial and tangential distortion parameters for the left and right cameras separately for a stereo pair.

If this is set, it will activate distortion mode, i.e. the image will be undistorted before passing to the SLAM tracker. NOTE: This must be called after setting the camera intrinsics.

Parameters

distortionCoefficientsLeft	Radial (k) and tangential (p) coefficients for the left camera, in the standard OpenCV order [k0, k1, p0, p1, k2, k3, k4, k5]
distortionCoefficientsRight	Radial (k) and tangential (p) coefficients for the right camera, in the standard OpenCV order [k0, k1, p0, p1, k2, k3, k4, k5]

Returns

A CameraParameterError indicating whether the distortion was set up correctly, and otherwise what went wrong

3.6.4.30 CameraParameterError KudanSLAM::setCameraRectification (KudanMatrix3 *intrinsicsLeft,* KudanMatrix3 *rotationLeft,* std::vector< float > *distortionCoefficientsLeft,* KudanMatrix3 *intrinsicsRight,* KudanMatrix3 *rotationRight,* std::vector< float > *distortionCoefficientsRight*)

Set the rectification extrinsics for a stereo camera pair, plus undistortion for each individual camera.

NOTE: This must be called after setting the camera intrinsics. Those intrinsics are treated as the intrinsics after rectification; the intrinsics before rectification (for each camera) must also be given.

Parameters

intrinsicsLeft	The intrinsics of the left camera prior to rectification (and undistortion)
rotationLeft	The rotation matrix of the left camera with respect to the stereo origin
distortionCoefficientsLeft	Distortion coefficients (radial and tangential) for the left camera
intrinsicsRight	The intrinsics of the right camera prior to rectification (and undistortion)
rotationRight	The rotation matrix of the right camera with respect to the stereo origin
distortionCoefficientsRight	Distortion coefficients (radial and tangential) for the right camera

Returns

A CameraParameterError indicating whether the distortion/rectification was set up correctly, and otherwise what went wrong

3.6.4.31 CameraParameterError KudanSLAM::setCameraRectification (float *fx_l,* float *fy_l,* float *cx_l,* float *cy_l,* KudanMatrix3 *rotationLeft,* std::vector< float > *distortionCoefficientsLeft,* float *fx_r,* float *fy_r,* float *cx_r,* float *cy_r,* KudanMatrix3 *rotationRight,* std::vector< float > *distortionCoefficientsRight*)

Set the rectification extrinsics for a stereo camera pair, plus undistortion for each individual camera.

NOTE: This must be called after setting the camera intrinsics. Those intrinsics are treated as the intrinsics after rectification; the intrinsics before rectification (for each camera) must also be given.

Parameters

fx_l	The focal length (x) of the left camera prior to rectification (and undistortion)
fy_l	The focal length (y) of the left camera prior to rectification (and undistortion)
cx_l	The principal point (x) of the left camera prior to rectification (and undistortion)
cy_l	The principal point (y) of the left camera prior to rectification (and undistortion)
rotationLeft	The rotation matrix of the left camera with respect to the stereo origin
distortionCoefficientsLeft	Distortion coefficients (radial and tangential) for the left camera
fx_r	The focal length (x) of the right camera prior to rectification (and undistortion)
fy_r	The focal length (y) of the right camera prior to rectification (and undistortion)
cx_r	The principal point (x) of the right camera prior to rectification (and undistortion)
cy_r	The principal point (y) of the right camera prior to rectification (and undistortion)
rotationRight	The rotation matrix of the right camera with respect to the stereo origin
distortionCoefficientsRight	Distortion coefficients (radial and tangential) for the right camera

Returns

A CameraParameterError indicating whether the distortion/rectification was set up correctly, and otherwise what went wrong

3.6.4.32 bool KudanSLAM::setDescriptorMode ( )

Set the SLAM system to use descriptors for tracking and expansion.

Returns

A boolean indicating whether this setting was set correctly

3.6.4.33 bool KudanSLAM::setMonocularMode ( )

Set that the SLAM system will be running in monocular mode.

Expect single-image input.

Returns

A boolean indicating whether this setting was set correctly

3.6.4.34 bool KudanSLAM::setPatchMode ( )

Set the SLAM system to use patches for tracking and expansion.

Returns

A boolean indicating whether this setting was set correctly

3.6.4.35 void KudanSLAM::setRequestCameraMatrices (bool *request*)

Request that the camera poses (represented as 4x4 matrices in camera and world space) should be stored on the next run.

Parameters

request	A request to store these data or not

3.6.4.36 void KudanSLAM::setRequestCameraPoses (bool *request*)

Request that the camera poses (represented as rotation matrices, quaternions and translations in camera and world space) should be stored on the next run.

Parameters

request	A request to store these data or not

3.6.4.37 void KudanSLAM::setRequestPoints2D (bool *request*)

Set whether 2D points are to be saved during the snapshot.

These are the map points and visible/tracked points, which will be available through the getMapPoints3D functions etc

Parameters

request	A request to store these data or not

3.6.4.38 void KudanSLAM::setRequestPoints3D (bool *request*)

Set whether 3D points are to be saved during the snapshot.

These are the map points and visible/tracked points, which will be available through the getMapPoints3D functions etc

Parameters

request	A request to store these data or not

3.6.4.39 void KudanSLAM::setShouldExpand (bool *shouldExpand*)

Request expansion now.

This should *not* be needed in usual SLAM, but remains to provide a way to specify manual expansion.

Parameters

shouldExpand	To set whether expansion should happen now. If set to true, SLAM will expand at the next opportunity; if false, it will cancel any pending expansion requests.

3.6.4.40 bool KudanSLAM::setStereoMode ( )

Set that the SLAM system will be running in stereo mode.

Expect stereo input.

Returns

A boolean indicating whether this setting was set correctly

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

- SLAM/Interface.hpp

3.7 KudanVector2 Class Reference

A simple representation of a 2D vector.

`#include <Interface.hpp>`

Public Attributes

- float x
  X coordinate.
- float y
  Y coordinate.

3.7.1 Detailed Description

A simple representation of a 2D vector.

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

- SLAM/Interface.hpp

3.8 KudanVector3 Class Reference

A simple representation of a 3D vector.

`#include <Interface.hpp>`

Public Attributes

- float x
  X coordinate.
- float y
  Y coordinate.
- float z
  Z coordinate.

3.8.1 Detailed Description

A simple representation of a 3D vector.

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

- SLAM/Interface.hpp

Index