Stereo SVO SLAM

Generated by Doxygen 1.8.13

Contents

Chapter 1

Stereo SVO SLAM Library

This repository contains a proof of concept for a SVO based stereo camera SLAM library.

The repository is organized as follows:

Direcotry	Description
doc	Documentation of the algorithm, implementation, etc.
src	Source code of library, test application, demo, qt viewer and python wrapper
test	Test scripts, test videos, etc.

The source code includes some doxygen comments. Check out the documentation folder for indepth information.

The thesis and source code documentation can be found here:

`Thesis Documentation`

`Source Documentation`

Library

The library allows to process stereo images and calculates the camera position based on this images.

Test application

The test application allows to process input images from different sources like Econ Tara Camera, EuRoC dataset or video input. It requires a YAML file with camera parameters. See src/app/Blender.yaml for more details.

Demo application

The demo application is a simple ar-application which shows what a SLAM library can do. It only supports Econ Tara an requires a YAML file with camera settings (src/app/Econ.yaml).

Qt 3D Viewer

The Qt 3D viewer can connect to the test application to show keyframes, current pose and trajectory.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

Class Documentation

4.1 CameraSettings Struct Reference

Structure representing the settings for the SLAM algorithm for a specific camera type.

`#include <stereo_slam_types.hpp>`

Public Attributes

- float baseline
  Baseline of the camera in meter times fx.
- float fx
  Focal length in pixels along x axis.
- float fy
  Focal length in pixels along y axis.
- float cx
  Camera principal point along x axis.
- float cy
  Camera principal point along y axis.
- float k1
  Distortion parameter k1 (radial 1)
- float k2
  Distortion parameter k2 (radial 2)
- float k3
  Distortion parameter k3 (radial 3)
- float p1
  Distortion parameter p1 (tangential 1)
- float p2
  Distortion parameter p2 (tangential 2)
- int grid_height
  Grid height in pixels (for keypoint detection)
- int grid_width
  Grid width in pixels (for keypoint detection)
- int search_x
  Depth calculator maximum disparity.

- int search_y
  Depth calculator maximum missalignment in y direction.
- int window_size_pose_estimator
  Window size for pose estimation (4 works okay)
- int window_size_opt_flow
  Window size for optical flow (31 works okay)
- int window_size_depth_calculator
  Window size for depth calculator (31 works okay)
- int max_pyramid_levels
  Maximum pyramid levels for pose estimation.
- int min_pyramid_level_pose_estimation
  Minimum pyramid level for pose estimation (e.g. if max =3 and min=2 it wont search on level 1)

4.1.1 Detailed Description

Structure representing the settings for the SLAM algorithm for a specific camera type.

Each camera works best with it's own settings. Some parameters like fx, fy, etc. are physicaly given, some are not (e.g. grid_height)

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

- include/stereo_slam_types.hpp

4.2 Color Struct Reference

Color representation.

`#include <stereo_slam_types.hpp>`

Public Attributes

- uint8_t r
  red
- uint8_t g
  green
- uint8_t b
  blue

4.2.1 Detailed Description

Color representation.

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

- include/stereo_slam_types.hpp

4.3 CornerDetector Class Reference

Detect corner points and edgleds (internal use)

`#include <corner_detector.hpp>`

Public Member Functions

- void **detect_keypoints** (const Mat &image, int grid_width, int grid_height, vector< KeyPoint2d > &keypoints, vector< KeyPointInformation > &kp_info, int level)

4.3.1 Detailed Description

Detect corner points and edgleds (internal use)

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

- include/corner_detector.hpp

4.4 DepthCalculator Class Reference

Class that estimates depth (internal use)

`#include <depth_calculator.hpp>`

Public Member Functions

- void calculate_depth (Frame &frame, const struct CameraSettings &camera_settings)

  Search corner points and calculate depth.

4.4.1 Detailed Description

Class that estimates depth (internal use)

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

- include/depth_calculator.hpp

4.5 DepthFilter Class Reference

Update point cloud, detect outliers (internal use)

`#include <depth_filter.hpp>`

Public Member Functions

- **DepthFilter** (KeyFrameManager &keyframe_manager, const CameraSettings &camera_settings)
- void update_depth (Frame &frame, std::vector< KeyPoint3d > &updated_kps3d)

  Update the 3D keypoints.

4.5.1 Detailed Description

Update point cloud, detect outliers (internal use)

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

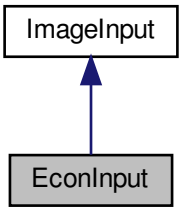
- include/depth_filter.hpp

4.6 EconInput Class Reference

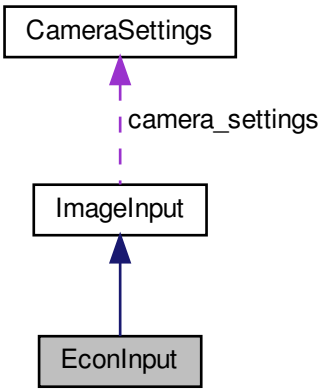
Class for Econ Tara video input.

`#include <econ_input.hpp>`

Inheritance diagram for EconInput:



Collaboration diagram for EconInput:



Public Member Functions

- EconInput (const std::string &camera_path, const std::string &hidraw_path, const std::string &settings, const std::string &hidraw_imu_path="")

  Create the EconInput object.
- virtual bool read (cv::Mat &left, cv::Mat &right, float &time_stamp)

  Read a new image from the camera including a timestamp.
- virtual void get_camera_settings (CameraSettings &camera_settings)

  Get the camera settings from the yaml file.
- virtual bool set_manual_exposure (int exposure)

  Set the exposure value (1 = auto exposure -> 30000)
- virtual bool configure_imu ()

  Configure IMU (e.g. frequency, resolution, etc.)
- virtual bool get_imu_data (ImuData &imu_data)

  Read IMU data.
- virtual bool set_hdr (bool hdr)

  Set HDR mode to on or off.
- virtual bool read_temperature (float &temperature)

  Read camera temperature.
- float **get_freqency** ()
- void calibrate_imu ()

  Calibrate the IMU.
- bool imu_available ()

  Check if IMU is available.

Additional Inherited Members

4.6.1 Detailed Description

Class for Econ Tara video input.

This class accepts a path to the video file, the hidraw device to control exposer etc, the YAML settings file and the

4.6.2 Constructor & Destructor Documentation

4.6.2.1 EconInput()

```
EconInput::EconInput (
            const std::string & camera_path,
            const std::string & hidraw_path,
            const std::string & settings,
            const std::string & hidraw_imu_path = "" )
```

Create the EconInput object.

Parameters

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

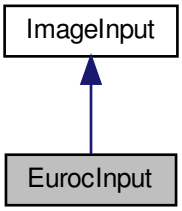
- app/econ_input.hpp

4.7 EurocInput Class Reference

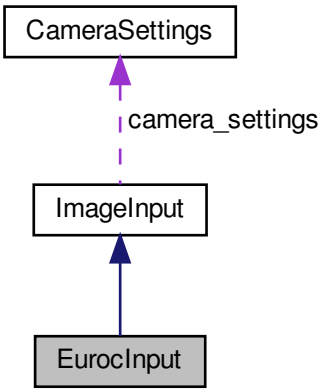
Class for euroc video input.

`#include <euroc_input.hpp>`

Inheritance diagram for EurocInput:



Collaboration diagram for EurocInput:



Public Member Functions

- EurocInput (const std::string &image_path, const std::string &settings)

  Create the EurocInput object.
- virtual bool read (cv::Mat &left, cv::Mat &right, float &time_stamp)

  Read a new image from the camera including a timestamp.
- virtual void get_camera_settings (CameraSettings &camera_settings)

  Get the camera settings from the yaml file.
- void jump_to (int frame_number)

  Jump to a specific frame.

Additional Inherited Members

4.7.1 Detailed Description

Class for euroc video input.

This class accepts a path to the euroc mav0 folder and then starts to read the images from there.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 EurocInput()

```
EurocInput::EurocInput (
            const std::string & image_path,
            const std::string & settings )
```

Create the EurocInput object.

Parameters

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

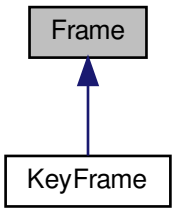
- app/euroc_input.hpp

4.8 Frame Struct Reference

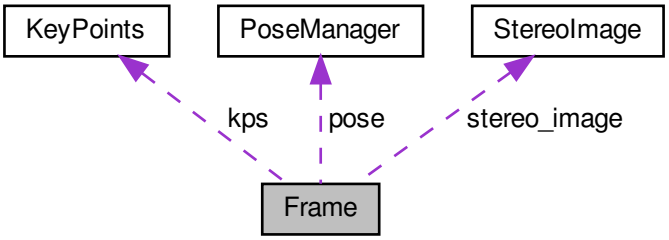
Representation of a frame.

```
#include <stereo_slam_types.hpp>
```

Inheritance diagram for Frame:



Collaboration diagram for Frame:



Public Attributes

- uint64_t id
  The frame id.
- PoseManager pose
  Pose assigned to the frame.
- struct StereoImage stereo_image
  Stereo image assigned to the framek.
- struct KeyPoints kps
  Keypoints used by the frame.
- double time_stamp
  Timestamp.

4.8.1 Detailed Description

Representation of a frame.

This includes pose, keypoints, stereo image, etc.

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

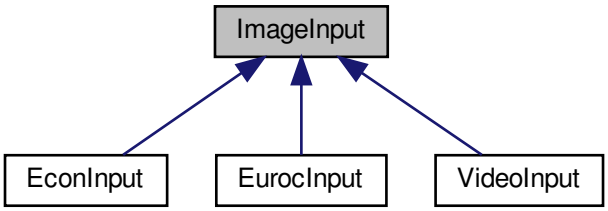
- include/stereo_slam_types.hpp

4.9 ImageInput Class Reference

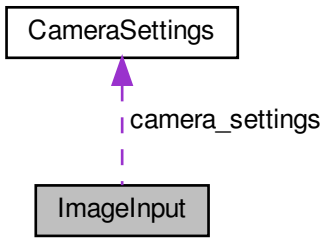
Abstract class for image input.

`#include <image_input.hpp>`

Inheritance diagram for ImageInput:



Collaboration diagram for ImageInput:



Public Member Functions

- virtual bool read (cv::Mat &left, cv::Mat &right, float &time_stamp)=0
  Read a new image from the camera including a timestamp.
- virtual void get_camera_settings (CameraSettings &camera_settings)=0
  Get the camera settings from the yaml file.
- uint32_t get_fps () const
  Get the frame rate of the camera.

Protected Member Functions

- virtual void **read_settings** (const std::string &settings)

Protected Attributes

- uint32_t **fps**
- CameraSettings **camera_settings**

4.9.1 Detailed Description

Abstract class for image input.

Abstract class used for all kind of input image sources. Sources can be videos, image series or cameras.

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

- app/image_input.hpp

4.10 ImuData Struct Reference

Data received from IMU.

`#include <econ_input.hpp>`

Public Attributes

- float acceleration_x
  Acceleration in x direction.
- float acceleration_y
  Acceleration in y direction.
- float acceleration_z
  Acceleration in z direction.
- float gyro_x
  Angle velocity around x axis.
- float gyro_y
  Angle velocity around y axis.
- float gyro_z
  Angle velocity around z axis.

4.10.1 Detailed Description

Data received from IMU.

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

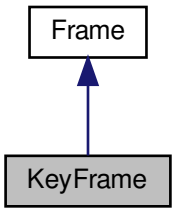
- app/econ_input.hpp

4.11 KeyFrame Struct Reference

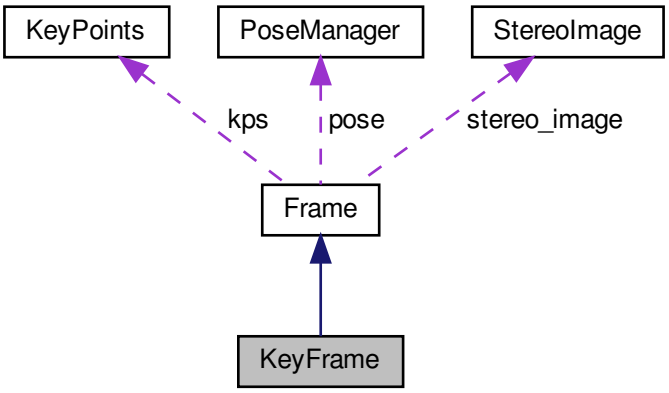
A keyframe is a frame which inserts new keypoints.

`#include <stereo_slam_types.hpp>`

Inheritance diagram for KeyFrame:



Collaboration diagram for KeyFrame:



Additional Inherited Members

4.11.1 Detailed Description

A keyframe is a frame which inserts new keypoints.

It's basically a normal frame with a new id. The algorithm decides when to insert a new frame.

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

- include/stereo_slam_types.hpp

4.12 KeyFrameManager Class Reference

Manages key frames (internal use only)

`#include <keyframe_manager.hpp>`

Public Member Functions

- **KeyFrameManager** (const CameraSettings &camera_settings)
- KeyFrame * create_keyframe (Frame &frame)

  Create a new keyframe.
- KeyFrame * get_keyframe (uint32_t id)

  Get the keyframe with id.
- void get_keyframes (std::vector< KeyFrame > &keyframes)

  Get the all keyframes.
- bool keyframe_needed (const Frame &frame)

  Is a new keyframe needed?

4.12.1 Detailed Description

Manages key frames (internal use only)

Decides when to insert new keyframes, allows to create new keyfames, etc.

4.12.2 Member Function Documentation

4.12.2.1 create_keyframe()

```
KeyFrame* KeyFrameManager::create_keyframe (
            Frame & frame )
```

Create a new keyframe.

Parameters

in	*frame*	The frame that should be used to create the keyframe

Returns

Pointer to the generated keyframe

4.12.2.2 get_keyframe()

```
KeyFrame* KeyFrameManager::get_keyframe (
            uint32_t id )
```

Get the keyframe with id.

Parameters

in	*id*	The keyframe id

Returns

A pointer to the keyframe

4.12.2.3 get_keyframes()

```
void KeyFrameManager::get_keyframes (
            std::vector< KeyFrame > & keyframes )
```

Get the all keyframes.

Parameters

out	*keyframes*	vector with keyframes

Returns

A pointer to the keyframe

4.12.2.4 keyframe_needed()

```
bool KeyFrameManager::keyframe_needed (
            const Frame & frame )
```

Is a new keyframe needed?

Parameters

in	*Current*	frame

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

- include/keyframe_manager.hpp

4.13 KeyPoint2d Struct Reference

A 2D keypoint representation in the image.

`#include <stereo_slam_types.hpp>`

Public Attributes

- float x
  X position.
- float y
  Y position.

4.13.1 Detailed Description

A 2D keypoint representation in the image.

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

- include/stereo_slam_types.hpp

4.14 KeyPoint3d Struct Reference

A 3D keypoint representation in global coordinates.

`#include <stereo_slam_types.hpp>`

Public Attributes

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

4.14.1 Detailed Description

A 3D keypoint representation in global coordinates.

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

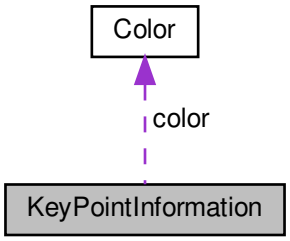
- include/stereo_slam_types.hpp

4.15 KeyPointInformation Struct Reference

Structure holding information about a keypoint.

`#include <stereo_slam_types.hpp>`

Collaboration diagram for KeyPointInformation:



Public Attributes

- float score

  Score achieved during keypoint detection.
- int level

  Pyramid level on which keypoint was detected.
- enum KeyPointType type

  The keypoint type.
- uint64_t keyframe_id

  The keyframe id where the keypoint was found.
- size_t keypoint_index

  The keypoint index in the keyframe.
- Color color

  The color of the keypoint (for debugging)
- bool ignore_during_refinement

  Should be ignored during refinement.
- bool ignore_completely

  Should be ignored completely (will be removed in next run)
- int outlier_count

  The count the point is rates as outlier so far.
- int inlier_count

  The count the point is rated as inlier so far.
- bool ignore_temporary

  Ignore the keypoint temporary.
- cv::KalmanFilter kf

  Kalman filter used for depth filter including parameters.

4.15.1 Detailed Description

Structure holding information about a keypoint.

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

- include/stereo_slam_types.hpp

4.16 KeyPoints Struct Reference

Collection of keypoints.

`#include <stereo_slam_types.hpp>`

Public Attributes

- std::vector< KeyPoint2d > kps2d
  2D keypoints
- std::vector< KeyPoint3d > kps3d
  3D keypoints
- std::vector< KeyPointInformation > info
  Information about the keypoint.

4.16.1 Detailed Description

Collection of keypoints.

Each entry has the same index. We try to avaoid mixing information to speed up calculation (e.g. kp2d<->kp3d mixing). The size of all elements should be the same.

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

- include/stereo_slam_types.hpp

4.17 OpticalFlow Class Reference

Wrapper for opencv optical flow (internal use)

`#include <optical_flow.hpp>`

Public Member Functions

- **OpticalFlow** (const CameraSettings &camera_settings)
- void calculate_optical_flow (const StereoImage &previous_stereo_image_pyr, const std::vector< KeyPoint2d > &previous_keypoints2d, const StereoImage &current_stereo_image_pyr, std::vector< KeyPoint2d > &current_keypoints2d, std::vector< float > &err)
  Calculate the optical flow.

4.17.1 Detailed Description

Wrapper for opencv optical flow (internal use)

4.17.2 Member Function Documentation

4.17.2.1 calculate_optical_flow()

```
void OpticalFlow::calculate_optical_flow (
            const StereoImage & previous_stereo_image_pyr,
            const std::vector< KeyPoint2d > & previous_keypoints2d,
            const StereoImage & current_stereo_image_pyr,
            std::vector< KeyPoint2d > & current_keypoints2d,
            std::vector< float > & err )
```

Calculate the optical flow.

Parameters

in	*previous_stereo_image_pyr*	Reference stereo image pyramid
in	*previous_keypoints2d*	Keypoints in the previous image
in	*current_stereo_image_pyr*	Stereo image pyramid to compare
in	*current_keypoints2d*	Keypoints in the image to compare
out	*err*	Intensity difference for each point

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

- include/optical_flow.hpp

4.18 Pose Struct Reference

Structure representing a global Pose in 3D.

```
#include <pose_manager.hpp>
```

Public Attributes

- float x
  X position in room.
- float y
  Y position in room.
- float z
  Z position in room.
- float rx

Rotation around x axis.

- float ry

  Rotation around y axis.

- float rz

  Rotation around z axis.

4.18.1 Detailed Description

Structure representing a global Pose in 3D.

We use the same right-handed-coordinate system as OpenCV.
positive x: move right
positive y: move down
positive z: move forward
This is not the same as an extrinsic camera matrix. We first rotate and then move. Also it is inverse to the extrinsic camera matrix. We need to do -position and then rotate with -rotation. Also note that first we rotate around z axis, then around y and finally around x. This may be different for describtion in Qt3D or robotoics. There it's often rotate around x, y, z. PoseManager can help in this case by providing a wrapper.

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

- include/pose_manager.hpp

4.19 PoseEstimator Class Reference

Class that does pose estimation based on sparse image alignment (internal use)

`#include <pose_estimator.hpp>`

Public Member Functions

- **PoseEstimator** (const StereoImage &current_stereo_image, const StereoImage &previous_stereo_image, const KeyPoints &previous_keypoints, const CameraSettings &camera_settings)
- float estimate_pose (const PoseManager &pose_manager_guess, PoseManager &estimaged_pose)

4.19.1 Detailed Description

Class that does pose estimation based on sparse image alignment (internal use)

4.19.2 Member Function Documentation

4.19.2.1 estimate_pose()

```
float PoseEstimator::estimate_pose (
            const PoseManager & pose_manager_guess,
            PoseManager & estimaged_pose )
```

Estimate the pose based on an initial guess

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

- include/pose_estimator.hpp

4.20 PoseManager Class Reference

Class for managing the pose.

```
#include <pose_manager.hpp>
```

Public Member Functions

- void set_pose (Pose &pose)

  Set the pose via Pose structure.
- void set_vector (cv::Vec6f &pose)

  Set the pose via OpenCV Vector.
- cv::Matx33f get_rotation_matrix () const

  Receive the rotation matrix.
- cv::Matx33f get_inv_rotation_matrix () const

  Receive inverse rotation matrix.
- cv::Vec3f get_translation () const

  Receive translation (position)
- cv::Vec3f get_angles () const

  Receive rotation/angles as OpenCV vector.
- cv::Vec3f get_robot_angles () const

  Receive rotation/angles applied in inverse order (rx∗ry∗rz instead rz∗ry∗rx)
- Pose get_pose () const

  Receive the pose as Pose.
- cv::Vec6f get_vector () const

  Receive the Pose as vector.

Friends

- std::ostream & operator<< (std::ostream &os, const PoseManager &pose)

  allow printing a Pose

4.20.1 Detailed Description

Class for managing the pose.

Also see Pose. The pose manager makes sure that we calculate several matrices (e.g. rotation) only once. This should help to speed up the calculations. It also provides other interfaces to set or get poses and allows to receive a robot angels instead of OpenCV angles (x,y,z instead of z,y,x).

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

- include/pose_manager.hpp

4.21 PoseRefiner Class Reference

Do pose refinement based on optical flow (internal use)

`#include <pose_refinement.hpp>`

Public Member Functions

- **PoseRefiner** (const CameraSettings &camera_settings)
- float refine_pose (KeyFrameManager &keyframe_manager, Frame &frame)

  Refine the pose.

4.21.1 Detailed Description

Do pose refinement based on optical flow (internal use)

Handle pose refinement by first doing optical flow and then minimizes the rerpojection error

4.21.2 Member Function Documentation

4.21.2.1 refine_pose()

```
float PoseRefiner::refine_pose (
            KeyFrameManager & keyframe_manager,
            Frame & frame )
```

Refine the pose.

Parameters

in	*keyframe_manager*	All keyframes
in	*frame*	The frame

Returns

re-projection error

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

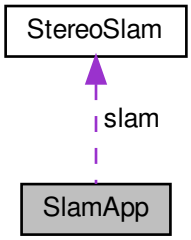
- include/pose_refinement.hpp

4.22 SlamApp Class Reference

Class which wraps camera handling, and StereoSlam.

`#include <slam_app.hpp>`

Collaboration diagram for SlamApp:



Public Member Functions

- bool initialize (const QString &camera_type, const QString &video, const QString &settings, const QString &trajectory_file=QString(), const QString &hidraw_settings=QString(), int exposure=1, bool hdr=false, int move=0, const QString &hidraw_imu=QString())

  Initalize the SLAM Applicaiton.
- bool start ()

  Start the SLAM App and all its threads.
- bool stop ()

  Stop the SLAM App and all its threads.
- void read_imu_data ()

  Read data from IMU.
- void read_image ()

  Read image from camera.
- bool process_image ()

  Process image.

Public Attributes

- StereoSlam * **slam**

4.22.1 Detailed Description

Class which wraps camera handling, and StereoSlam.

This class allows us to have a shared code base between e.g. ar-app and app.

Example:

```
SlamApp slam_app;
slam_app.initialize("econ", "/dev/video0", "Econ.yaml", "trajectory_out.csv",
         "/dev/hidraw0", 10000, false, 0, "/dev/hidraw1");
slam_app.start();
while (slam_app.process_image());

vector<Pose> trajectory;
slam_app.slam->get_trajectory(trajectory);
```

4.22.2 Member Function Documentation

4.22.2.1 initialize()

```
bool SlamApp::initialize (
             const QString & camera_type,
             const QString & video,
             const QString & settings,
             const QString & trajectory_file = QString(),
             const QString & hidraw_settings = QString(),
             int exposure = 1,
             bool hdr = false,
             int move = 0,
             const QString & hidraw_imu = QString() )
```

Initalize the SLAM Applicaiton.

Parameters

in	*camera_type*	Can be econ, euroc or video
in	*video*	The path to the video file or device
in	*trajecotry_file*	Where to store the trajectory (empty-> don't save)
in	*hidraw_settings*	Path to hidraw device for Settings
in	*expusre*	Exposure for econ camera
in	*hdr*	Enable/Disable HDR for econ
in	*move*	Skip n frames for video or EuRoC input
in	*hidraw_imu*	The hidraw device to receive IMU data from

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

- app/slam_app.hpp

4.23 StereoImage Struct Reference

Structure used to store a stereo image (internal use)

`#include <stereo_slam_types.hpp>`

Public Attributes

- std::vector< cv::Mat > left
  Left image pyramid (max_pyramid_levels), left[0] is the orignal.
- std::vector< cv::Mat > right
  Right image pyramid (currently only one level!), right[0] is the original.
- std::vector< cv::Mat > opt_flow
  Optical flow pyramid of left image used only for optical flwo (different from other pyramids)

4.23.1 Detailed Description

Structure used to store a stereo image (internal use)

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

- include/stereo_slam_types.hpp

4.24 StereoSlam Class Reference

Class for the whole SVO Stereo SLAM.

`#include <stereo_slam.hpp>`

Public Member Functions

- StereoSlam (const CameraSettings &camera_settings)
  Create the stereo SLAM object.
- void new_image (const cv::Mat &left, const cv::Mat &right, const float time_stamp)
  Process new image from the camera.
- void get_keyframe (KeyFrame &keyframe)
  Receive last keyframe.
- void get_keyframes (std::vector< KeyFrame > &keyframes)
  Receive all keyframes.
- bool get_frame (Frame &frame)
  Recevie frame.
- void get_trajectory (std::vector< Pose > &trajectory)
  Receive trajectory for all frames.
- Pose update_pose (const Pose &pose, const cv::Vec6f &speed, const cv::Vec6f &pose_variance, const cv::Vec6f &speed_variance, double dt)
  Update the camera pose externally (e.g. trought IMU)

4.24.1 Detailed Description

Class for the whole SVO Stereo SLAM.

This is the class used to create a SVO Stereo SLAM object. It is the main class used to interact with in user programs. Creating the object requires valid camera settings. After that it can be fed with new images.

Example:

```
StereoSlam slam(camera_settings);
slam.new_image(left, right, time_stamp);

vector<Pose> trajectory;
slam.get_trajectory(trajectory);
```

4.24.2 Constructor & Destructor Documentation

4.24.2.1 StereoSlam()

```
StereoSlam::StereoSlam (
            const CameraSettings & camera_settings )
```

Create the stereo SLAM object.

Parameters

in	*camera_settings*	The camera settings that define the camera in use

4.24.3 Member Function Documentation

4.24.3.1 new_image()

```
void StereoSlam::new_image (
            const cv::Mat & left,
            const cv::Mat & right,
            const float time_stamp )
```

Process new image from the camera.

Parameters

in	*left*	Left stereo image (from front of the camera)
in	*right*	Right stereo image (from front of the camera)
in	*time_stamp*	Time of when the image was taken in seconds (float)

4.24.3.2 update_pose()

```
Pose StereoSlam::update_pose (
            const Pose & pose,
            const cv::Vec6f & speed,
            const cv::Vec6f & pose_variance,
            const cv::Vec6f & speed_variance,
            double dt )
```

Update the camera pose externally (e.g. trought IMU)

Parameters

in	*pose*	A vector describeing the measured pose
in	*speed*	A vector describeing the measured speed (velocity) in x,y,z and rx,ry,rz direction
in	*pose_variance*	The variance of the pose measurement
in	*speed_variance*	The variance of the speed measurement
in	*dt*	Time since last update (1/f)

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

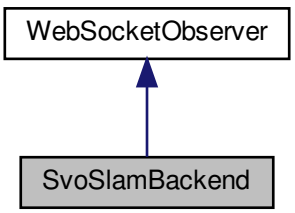
- include/stereo_slam.hpp

4.25 SvoSlamBackend Class Reference

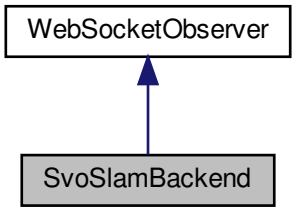
SVO SLAM Backend for Qt Viewer.

```
#include <svo_slam_backend.hpp>
```

Inheritance diagram for SvoSlamBackend:



Collaboration diagram for SvoSlamBackend:



Public Member Functions

- **SvoSlamBackend** (StereoSlam ∗slam)
- void text_message_received (QWebSocket &socket, const QString &message)

  Read text message and geenerate answer.

4.25.1 Detailed Description

SVO SLAM Backend for Qt Viewer.

This Class provides the data from StereoSlam to a WebSocket interface. It expects a SLAM object and must be register at the WebSocket server.

4.25.2 Member Function Documentation

4.25.2.1 text_message_received()

```
void SvoSlamBackend::text_message_received (
            QWebSocket & socket,
            const QString & message )  [virtual]
```

Read text message and geenerate answer.

This function reacts on messages from a client and generates an answer

Implements WebSocketObserver.

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

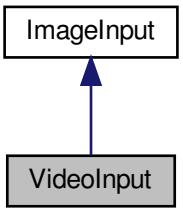
- app/svo_slam_backend.hpp

4.26 VideoInput Class Reference

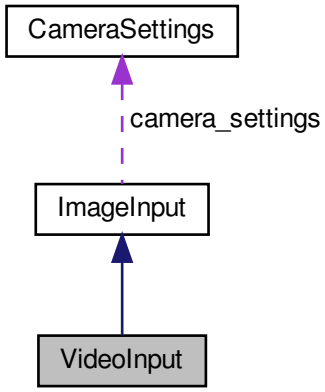
Class for Video File input.

`#include <video_input.hpp>`

Inheritance diagram for VideoInput:



Collaboration diagram for VideoInput:



Public Member Functions

- **VideoInput** (const std::string &video_path, const std::string &settings)
- virtual bool read (cv::Mat &left, cv::Mat &right, float &time_stamp)

  Read a new image from the camera including a timestamp.
- virtual void get_camera_settings (CameraSettings &camera_settings)

  Get the camera settings from the yaml file.
- void **jump_to** (int frame_number)

Additional Inherited Members

4.26.1 Detailed Description

Class for Video File input.

Video Files need to have the images aligned horizontally | video left | video right |

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

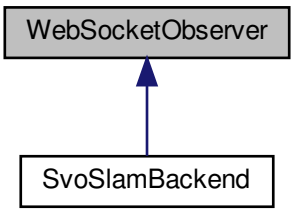
- app/video_input.hpp

4.27 WebSocketObserver Class Reference

WebSocket observer class.

`#include <websocketserver.hpp>`

Inheritance diagram for WebSocketObserver:



Public Member Functions

- virtual void text_message_received (QWebSocket &socket, const QString &message)=0
  Receive text message.

4.27.1 Detailed Description

WebSocket observer class.

Abstract class that should be inherited to write a WebSocketServer observer. If a meassage is received text_↩ message_received is called by WebSocketServer

4.27.2 Member Function Documentation

4.27.2.1 text_message_received()

```
virtual void WebSocketObserver::text_message_received (
            QWebSocket & socket,
            const QString & message )  [pure virtual]
```

Receive text message.

Receive a text message. We can use socket to send back an answer.

Parameters

in	*socket*	The socket where a message was received from
in	*message*	The message we received

Implemented in SvoSlamBackend.

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

- app/websocketserver.hpp

4.28 WebSocketServer Class Reference

Wrapper Class for QWebSocketServer.

`#include <websocketserver.hpp>`

Public Member Functions

- WebSocketServer (const char *server_name, int port, WebSocketObserver &observer, QWebSocket↩ Server::SslMode mode=QWebSocketServer::NonSecureMode)

  Create WebSocketServer.

4.28.1 Detailed Description

Wrapper Class for QWebSocketServer.

This class provides an easy to use weboscket server implementation. Every class interested in messages from the WebSocketServer should implement WebSocketObserver and register itself in the WebSocketerServer.

4.28.2 Constructor & Destructor Documentation

4.28.2.1 WebSocketServer()

```
WebSocketServer::WebSocketServer (
            const char * server_name,
            int port,
            WebSocketObserver & observer,
            QWebSocketServer::SslMode mode = QWebSocketServer::NonSecureMode )
```

Create WebSocketServer.

Create a websocker server and register an observer

Parameters

in	*server_name*	The name of the server (can be anything)
in	*port*	The port to listen on
in	*observer*	An observer that is called when a message arrives
in	*mode*	If SSL should be used or not

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

- app/websocketserver.hpp

Index