ZED

SDK

v..

Contents

Release Notes

ZED SDK 1.0.0

SDK

∗**NEW**∗ Main Features

- Positional Tracking : We introduce in 1.0.0 a whole new possibility of using the ZED as a positionnal tracker, along with depth estimation. Already shown in ZEDfu app, we now give the way for developers to create their own application with depth and pose estimation provided by the ZED SDK.
- SVO compression (66% memory less) : SVO files integrate lossless compression that reduces the size by 66%.

Breaking Compatibility

- Camera::init(...) : replace arguments by InitParams structure. See SDK Changelog.
- SENSING_MODE : new names to better reflect the usage. Old names has been removed.
- VGA resolution, the new ZED firmware (1142) introduced Wide VGA, the resolution is now 672∗376 for each image with a wider fielf of view. Update of new firmware is automatically done during installation of the ZED SDK.

SDK Changelog

- Added "InitParams" structure, including new parameters :

  o UNIT : all the data in metric will be scaled to fit the specified unit. The affected data are : baseline, depth, XYZ, depthclamp, closestDephValue, tracking information ... Therefore, "setBaselineRatio()" has been removed.

  o COORDINATE_SYSTEM : the oriented data such as (XYZ) point cloud and tracking information will be expressed in the desired coordinates system.

  o minimumDistance : the user can specify (under certain limits) the minimumDistance for depth computation, this will speed up the computation framerate.
- Added MODE::MEDIUM : this mode is close to the old PERFORMANCE, while PERFORMANCE has been speeded up.
- Unvalid MEASURE values are now separated into three values :

  o OCCLUSION_VALUE : NaN (not a number), for occlusions

  o TOO_FAR : Inf (infinity), for values above DepthClampValue

  o TOO_CLOSE : -Inf, for values below ClosestDepthValue

  o isValidMeasure() : macro to handle these values, return false if the given value is one of the above, else return true.
- Added support for ZED_EXPOSURE in ZEDCamera_settings structure. Exposure time can now be configured manually by the user.
- Added attribute for field of view in CamParameters structure. FoV of Left/Right images given by retrieveImage() are given in the new getParameters().LeftCam/RightCam.hFOV/dFOV/vFOV. See StereoParameters structure for more details.
- SVO files now integrates the global framerate as well as the time stamp, saved during the record process. Use the getCameraTimestamp() function to get the timestamp of the SVO current image. This can be used to detect frame drops during recording.

Samples

- Added two tracking samples, cpu/Simple Tracking and cpu/Tracking Viewer. Grabbing thread has be renamed Optimized Grab.
- Updated all samples

Dependencies

- Updated OpenCV, from 2.4.9 to 3.1 on Window / Linux. Jetson keeps the OpenCV4Tegra version, provided by NVIDIA . OpenCV3.1 is available in the dependencies folder under Windows.
- Added Eigen 3

Tools

- new unified GUI
- ZED Explorer:

  o Firmware update of the ZED now available on every platform (Win, Linux, Jetson).

ZED SDK 0.9.4 Beta

Platforms

- Linux : Added compatibility for Linux Kernel 4.X. Previous ZED SDK version were incompatible with new Linux Kernel, producing a "ZED_NOT_AVAILABLE" message and segfault.

SDK

- Fix a bug in color correction.

Tools

- Introducing a new tool called "ZED Calibration" that can be used to re-calibrate easily your ZED. Must be used in Live mode. Please follow the tutorial inside the tool to know how to use this new tool.
- Added automatic check of up-to-date ZED SDK version in ZED Explorer . Use ZED Explorer with -v command line option to check if your SDK is the last version available. This check is also done each time ZED Explorer is launched without command line options.
- Bug fix in ZED Depth Viewer that could lead to bad performances on specific CPU/GPU configuration on Windows.

App

- Introducing on Windows platform a new App called "ZEDfu". You can find this app in the new folder called "App". This App shows the power of the ZED to be able to reconstruct 3D model and mesh of an environment or an area. Long-awaited by many of you, we are finally releasing the first beta version of this App.
- The "App" folder will contain more App/softwares in next releases that will show more of the power of the ZED and the ZED SDK.

ZED SDK 0.9.3 Beta

Platforms

- Compatible with CUDA 7.5 for Windows and Linux.

SDK

- Improved grab() time.
- Reduce Latency when using cpu functions for Jetson.
- Moved Selfcalibration in a non-blocking function. Self-Calibration is done in background and the result can be checked with the status function.
- Improved Init() time

Tools

- Merged ZED Explorer and ZED Settings App into one single tool.
- Added command line options for ZED Explorer for check compatibility and recording

Known Issues

- On Ubuntu 14.04, the newer kernel 4.X is NOT supported.

ZED SDK 0.9.2b Beta

SDK

Bug fixes. Patches version.

ZED SDK 0.9.2 Beta

Platforms

- Added Window 10 compatibility.

SDK

- Improved Left/Right image quality with retrieveImage∗().
- Improved grab time on embedded platform (Jetson TK1).
- Added possibility to disable self-calibration during init().
- Added function that returns results of self-calibration.
- More calibration parameters available.

Tools

- Bug fixes in Depth Viewer tool.
- Added more calibration parameters in ZED Settings App

ZED SDK 0.9.1 Beta

SDK

- Bug fixes for Auto-calibration that made samples crash on Windows 8
- Bug fixes for ply write function that made incoherent color.
- Add more support for slMat2cvMat function.

Tools

- Improved global framerate of Depth Viewer tool.

ZED SDK 0.9.0 Beta

SDK

- Added triangulation function to extract XYZ or XYZRGBA point cloud.
- Added Save functions to save point cloud in multiple formats and depth image in multiple format.
- Added Camera and Current Timestamp extraction function to help synchronization with other devices.
- Added FPS extraction function.
- Bug fixes when using svo files.
- Improved AutoCalibration.

Samples

- Complete re-factory of samples.
- Added Cuda sample for background substraction and image disparity for right image.
- Added ROS Sample to demonstrate how to interact with ROS.
- Added Recording sample to demonstrate how to use recording functions.
- Added SVO Playback sample to demonstrate how to simply use the svo file.
- Simplify existing samples.

Tools

- Added ZED Depth Viewer sample to have a plug and play tool to demonstrate all the functions of the ZED SDK.
- Added Cmd Line mode for ZED Explorer to record svo without Graphics.

ZED SDK 0.8.2 Beta

SDK

- Added flip mode (to work with ZED Camera vertically flipped)
- Added multiple input possibility under Linux system.

Samples

- Added Multiple input sample.

ZED SDK 0.8.1 Beta

SDK

- Major improvement of Disparity estimation in untextured areas and repetitive patterns
- Reduced grab() time by 10%
- Reduced retrieveImage() time to <0.5ms when grab is done with disparity enabled
- Added new function to adjust ZED camera parameters (set / getCameraSettingsValue), including exposure, white balance, brightness, contrast and hue
- Added option to select a different framerate other than the default ones available in the ZED Camera constructor. For example, it is now possible to select VGA mode @ 30 fps
- Added getSVOPosition() to retrieve current SVO position
- Bug fix in 'Live mode' on Windows x64
- Added compatibility with NVIDIA GeForce 9 series of graphics cards

Tools

- Minor bug fix in ZED Explorer on Linux

ZED SDK 0.8.0 Beta

Platforms

- Added Linux (Ubuntu 14.04 LTS only) compatibility.

Tools

- Added SVOEditor: command line tool to cut and concatenate SVO files. See -help for more details
- Bug fix in ZED Settings App with "-sn" options
- Minor bug fix in ZED Explorer

SDK

- Improved Disparity estimation
- Bug fix in Disparity/Depth normalization
- Bug fix in NONE grab mode (MODE::NONE)
- Merged MODE::MEDIUM et MODE::QUALITY into a single QUALITY MODE
- Add specific functions for SVO (set position and get number of frames)
- Changed name of "DispReliability" to "ConfidenceThreshold"
- Conformed set/get functions

Known Issues

- Tools or Samples may not launch when run twice on Win8. Reconnecting the ZED will solve this issue.

ZED SDK 0.7.1a Alpha

Notes

- Initial release.

Known Issues

- Specific samples can crash at launch on Win 8. Relaunching the sample solves the problem.

Module Index

Modules

Here is a list of all modules:

Namespace Index

sl Namespace Reference

Namespaces

- zed

Classes

- struct uchar3Struct
- struct float3Struct
- struct double3Struct

Typedefs

- typedef unsigned char uchar
- typedef struct sl::uchar3Struct uchar3
- typedef struct sl::float3Struct float3
- typedef struct sl::double3Struct double3

Enumerations

- enum DEPTH_FORMAT { PNG, PFM, PGM, LAST_DEPTH_FORMAT }

  Enumerate the file format for depth saving.
- enum POINT_CLOUD_FORMAT {
  XYZ, PCD, PLY, VTK,
  LAST_CLOUD_FORMAT }

  Enumerate the file format avaliable for saving point clouds (depth triangulation)

Functions

- SLSTEREO_EXPORT_DLL bool writeDepthAs (sl::zed::Camera ∗zed, sl::DEPTH_FORMAT format, std::string name, float factor=1.)

  write the current depth map into a file
- SLSTEREO_EXPORT_DLL bool writePointCloudAs (sl::zed::Camera ∗zed, sl::POINT_CLOUD_FORMAT format, std::string name, bool withColor=false, bool keepOccludedPoint=false)

write the current point cloud into a file

Typedef Documentation

typedef struct sl::double3Struct double3

typedef struct sl::float3Struct float3

typedef unsigned char uchar

typedef struct sl::uchar3Struct uchar3

Function Documentation

SLSTEREO_EXPORT_DLL bool sl::writeDepthAs (sl::zed::Camera * *zed*, sl::DEPTH_FORMAT *format*, std::string *name*, float *factor = 1.*)

write the current depth map into a file

Parameters

DEPTH_FORMAT	: for file format
name	: the name (path) in which the depth will be saved
factor	: only for PNG and PGM, apply a gain to the depth value (default 1.) The maximum value is 65536, so you can set the Camera::setDepthClampValue to 20000 and give a factor to 3, Do not forget to scale (by 1./factor) the pixel value to get the real depth. The occlusions are represented by 0.

Return values

return	false if something wrong happen, else return true

Note

factor : only for PNG and PGM

SLSTEREO_EXPORT_DLL bool sl::writePointCloudAs (sl::zed::Camera * *zed*, sl::POINT_CLOUD_FORMAT *format*, std::string *name*, bool *withColor = `false`*, bool *keepOccludedPoint = `false`*)

write the current point cloud into a file

Parameters

POINT_CLOUD_FORMAT	: for file format
name	: the name (path) in which the point cloud will be saved
withColor	: indicates if the color must be saved (default false). Not avaliable for XYZ and VTK
keepOccludedPoint	: indicates if the non avaliable data should be saved and set to 0 (default false), if set to true this give a Point Cloud with a size = height * width

Return values

return	false if something wrong happen, else return true

Note

The color is not saved for XYZ and VTK files

Class Index

Class List

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

Module Documentation

Public enumeration types

Enumerations

- enum DEPTH_FORMAT { PNG, PFM, PGM, LAST_DEPTH_FORMAT }

  Enumerate the file format for depth saving.
- enum POINT_CLOUD_FORMAT {
  XYZ, PCD, PLY, VTK,
  LAST_CLOUD_FORMAT }

  Enumerate the file format avaliable for saving point clouds (depth triangulation)

- enum MODE {
  NONE, PERFORMANCE, MEDIUM, QUALITY,
  LAST_MODE }

  Enumerate for the pre-defined disparity map computation parameters.
- enum VIEW_MODE {
  STEREO_LEFT, STEREO_RIGHT, STEREO_ANAGLYPH, STEREO_DIFF,
  STEREO_SBS, STEREO_OVERLAY, STEREO_LEFT_GREY, STEREO_RIGHT_GREY,
  LAST_VIEW_MODE }

  Enumerate for available output views for monitoring.
- enum SIDE {
  LEFT, RIGHT, LEFT_GREY, RIGHT_GREY,
  LAST_SIDE }

  Enumerate for the Left and Right side of stereo Camera.
- enum SENSING_MODE { FILL, STANDARD, LAST_SENSING_MODE }

  Enumerate for the different types of disparity map computation.
- enum MEASURE {
  DISPARITY, DEPTH, CONFIDENCE, XYZ,
  XYZRGBA, LAST_MEASURE }

  Enumerate for the retrievable measure (each measure should be normalized to be displayed)
- enum ERRCODE {
  SUCCESS, NO_GPU_COMPATIBLE, NOT_ENOUGH_GPUMEM, ZED_NOT_AVAILABLE,
  ZED_RESOLUTION_INVALID, ZED_SETTINGS_FILE_NOT_AVAILABLE, INVALID_SVO_FILE, REC-
  ORDER_ERROR,
  INVALID_COORDINATE_SYSTEM, ZED_WRONG_FIRMWARE, LAST_ERRCODE }

  Enumerate for error code during the sl::zed::Camera::init.
- enum ZED_SELF_CALIBRATION_STATUS { SELF_CALIBRATION_NOT_CALLED, SELF_CALIBRAT-
  ION_RUNNING, SELF_CALIBRATION_FAILED, SELF_CALIBRATION_SUCCESS }

  Status for self calibration. Since v0.9.3, self-calibration is done in background and start in the Init or Reset function.
- enum ZEDResolution_mode {
  HD2K, HD1080, HD720, VGA,
  LAST_RESOLUTION }

  Enumerate for available resolutions for ZED camera.
- enum ZEDCamera_settings {
  ZED_BRIGHTNESS, ZED_CONTRAST, ZED_HUE, ZED_SATURATION,
  ZED_GAIN, ZED_EXPOSURE, LAST_SETTINGS }

  Enumerate the available camera settings for the ZED camera (contrast, hue, saturation, gain...)
- enum TRACKING_STATE {
  TRACKING_RELOCDB_INIT, TRACKING_INIT, TRACKING_RELOC, TRACKING_LOST,
  TRACKING_GOOD, TRACKING_OFF, TRACKING_LAST }

  Enumerate the current state of the tracking.
- enum TRACKING_FRAME_STATE { TRACKING_FRAME_NORMAL, TRACKING_FRAME_KEYFRA-
  ME, TRACKING_FRAME_CLOSE, TRACKING_FRAME_LAST }

  Enumerate the current state of the tracked frame.
- enum MAT_TRACKING_TYPE { PATH, POSE }

  Define which type of position matrix is used to store the tracking position.
- enum UNIT {
  MILLIMETER, METER, INCH, FOOT,
  LAST_UNIT }

  Enumerate for available metric unit of the depth.
- enum COORDINATE_SYSTEM {
  IMAGE = 1, LEFT_HANDED = 2, RIGHT_HANDED = 4, ROBOTIC = 8,
  APPLY_PATH = 16, LAST_COORDINATE_SYSTEM = 32 }

  Enumerate for available coordinates systems available for the points cloud.

Detailed Description

Enumeration Type Documentation

enum COORDINATE_SYSTEM

Enumerate for available coordinates systems available for the points cloud.

Enumerator

IMAGE

LEFT_HANDED

RIGHT_HANDED

ROBOTIC

APPLY_PATH

LAST_COORDINATE_SYSTEM

enum DEPTH_FORMAT

Enumerate the file format for depth saving.

Enumerator

PNG image format, in 16bits, the color encoding the depth

PFM stream of bytes, graphic image file format

PGM grayscale image format

LAST_DEPTH_FORMAT

enum ERRCODE

Enumerate for error code during the sl::zed::Camera::init.

Enumerator

SUCCESS Every step went fine, the camera is ready to use

NO_GPU_COMPATIBLE No GPU found or the CUDA cupability of the device is not sufficient

NOT_ENOUGH_GPUMEM not enough GPU memory for this depth calculation mode please try a "lighter" mode (PERFORMANCE...)

ZED_NOT_AVAILABLE The ZED camera is not plugged or detected

ZED_RESOLUTION_INVALID For Jetson only, resolution not yet supported (USB3.0 bandwidth)>

ZED_SETTINGS_FILE_NOT_AVAILABLE ZED Settings file is not found on the host machine. Use ZED Settings App tool to correct it.

INVALID_SVO_FILE The provided SVO file is not valid

RECORDER_ERROR An recorder related error occured (not enough free storage, invalid file)

INVALID_COORDINATE_SYSTEM The requested coordinate systeme is not available.

ZED_WRONG_FIRMWARE The firmware of the ZED is out of date. You might install the new version.

LAST_ERRCODE

enum MAT_TRACKING_TYPE

Define which type of position matrix is used to store the tracking position.

Enumerator

PATH The matrix contains the displacement from the first camera to the current one ($Path_n = Path_{n-1} * Pose_n^{-1}$)

POSE The matrix contains the displacement from the previous camera position to the current one

enum MEASURE

Enumerate for the retrievable measure (each measure should be normalized to be displayed)

Enumerator

DISPARITY Disparity map, 1 channel, FLOAT

DEPTH Depth map, 1 channel, FLOAT

CONFIDENCE Certainty/confidence of the diparity map, 1 channel, FLOAT

XYZ 3D coordinates of the image points, 4 channels, FLOAT (the 4th channel may contains the colors)

XYZRGBA 3D coordinates and Color of the image , 4 channels, FLOAT (the 4th channel encode 4 UCHAR for color)

LAST_MEASURE

enum MODE

Enumerate for the pre-defined disparity map computation parameters.

Enumerator

NONE Disparity map not computed, only the rectified images will be available.

PERFORMANCE Fastest mode, also requires less GPU memory, the disparity map is less robust

MEDIUM Balanced quality mode, requires less GPU memory but the disparity map is a little less detailed

QUALITY Better quality mode, the disparity map is more precise

LAST_MODE

enum POINT_CLOUD_FORMAT

Enumerate the file format avaliable for saving point clouds (depth triangulation)

Enumerator

XYZ 3D format, store the 3D coordinates

PCD Point Cloud Data file, store the 3D coordinates and the color

PLY PoLYgon file format, store the 3D coordinates and the color

VTK Visualization ToolKit file, store the 3D coordinates

LAST_CLOUD_FORMAT

enum SENSING_MODE

Enumerate for the different types of disparity map computation.

Enumerator

FILL Occlusion filling, edge sharpening, advanced post-filtering. Application example : Refocusing, Multi-view generation

STANDARD Structure conservative, no occlusion filling. Application example : Obstacle detection, 3D reconstructions, 3D measures

LAST_SENSING_MODE

enum SIDE

Enumerate for the Left and Right side of stereo Camera.

Enumerator

LEFT

RIGHT

LEFT_GREY

RIGHT_GREY

LAST_SIDE

enum TRACKING_FRAME_STATE

Enumerate the current state of the tracked frame.

Enumerator

TRACKING_FRAME_NORMAL Not a keyframe, normal behavior

TRACKING_FRAME_KEYFRAME The tracking detect a new reference image

TRACKING_FRAME_CLOSE The tracking find a previously known area and optimize the trajectory

TRACKING_FRAME_LAST

enum TRACKING_STATE

Enumerate the current state of the tracking.

Enumerator

TRACKING_RELOCDB_INIT has load a area database and is looking for a match

TRACKING_INIT relocalization initialization phase

TRACKING_RELOC The tracking is lost and try to relocate at a previously known position

TRACKING_LOST The tracking failed to relocate and is lost, a reset tracking should be performed

TRACKING_GOOD The tracking operates normally, the path should be correct

TRACKING_OFF The tracking is not enabled

TRACKING_LAST

enum UNIT

Enumerate for available metric unit of the depth.

Enumerator

MILLIMETER

METER

INCH

FOOT

LAST_UNIT

enum VIEW_MODE

Enumerate for available output views for monitoring.

Enumerator

STEREO_LEFT Left view

STEREO_RIGHT Right view

STEREO_ANAGLYPH Anaglyph (Red & Blue) view

STEREO_DIFF View of the difference between the left image and right image in gray scale

STEREO_SBS Side by Side view (in single image resolution)

STEREO_OVERLAY View of both images in 5O% transparency

STEREO_LEFT_GREY Left view grey scale

STEREO_RIGHT_GREY Right view grey scale

LAST_VIEW_MODE

enum ZED_SELF_CALIBRATION_STATUS

Status for self calibration. Since v0.9.3, self-calibration is done in background and start in the Init or Reset function.

You can follow the current status for the self-calibration any time once ZED object has been construct.

Enumerator

SELF_CALIBRATION_NOT_CALLED Self Calibration has not yet been called (no Camera::init or Camera::resetSelfCalibration called)

SELF_CALIBRATION_RUNNING Self Calibration is currently running.

SELF_CALIBRATION_FAILED Self Calibration has finished running but did not manage to get coherent values. Old Parameters are taken instead.

SELF_CALIBRATION_SUCCESS Self Calibration has finished running and did manage to get coherent values. New Parameters are taken.

enum ZEDCamera_settings

Enumerate the available camera settings for the ZED camera (contrast, hue, saturation, gain...)

Each enum defines one of those settings

Enumerator

ZED_BRIGHTNESS Defines the brightness control. Affected value should be between 0 and 8

ZED_CONTRAST Defines the constral control. Affected value should be between 0 and 8

ZED_HUE Defines the hue control. Affected value should be between 0 and 11

ZED_SATURATION Defines the saturation control. Affected value should be between 0 and 8

ZED_GAIN Defines the gain control. Affected value should be between 0 and 100 for manual control. If ZED_EXPOSURE is set to -1, the gain is in auto mode too.

ZED_EXPOSURE Defines the exposure control. A -1 value enable the AutoExposure/AutoGain control. Affected value should be between 0 and 100 for manual control. A 0 value only disable auto mode without modifing the last auto values, while a 1 to 100 value disable auto mode and set exposure to chosen value

LAST_SETTINGS

enum ZEDResolution_mode

Enumerate for available resolutions for ZED camera.

Warning

Since v1.0 the VGA mode is wider (from 640∗480 to 672∗376) and requires a firmware update to operate (>= 1142). It can be done using the ZED Explorer tool

Enumerator

HD2K 2208∗1242, supported framerate : 15 fps (unsupported by the Jetson TK1 at the moment)

HD1080 1920∗1080, supported framerates : 15, 30 fps (unsupported by the Jetson TK1 at the moment)

HD720 1280∗720, supported framerates : 15, 30, 60 fps, Jetson TK1 : 15 fps

VGA 672∗376, supported framerates : 15, 30, 60, 100 fps, Jetson TK1 : 15, 30 fps

LAST_RESOLUTION

Namespace Documentation

sl Namespace Reference

Namespaces

- zed

Classes

- struct uchar3Struct
- struct float3Struct
- struct double3Struct

Typedefs

- typedef unsigned char uchar
- typedef struct sl::uchar3Struct uchar3
- typedef struct sl::float3Struct float3
- typedef struct sl::double3Struct double3

Enumerations

- enum DEPTH_FORMAT { PNG, PFM, PGM, LAST_DEPTH_FORMAT }

  Enumerate the file format for depth saving.
- enum POINT_CLOUD_FORMAT {
  XYZ, PCD, PLY, VTK,
  LAST_CLOUD_FORMAT }

  Enumerate the file format avaliable for saving point clouds (depth triangulation)

Functions

- SLSTEREO_EXPORT_DLL bool writeDepthAs (sl::zed::Camera ∗zed, sl::DEPTH_FORMAT format, std::string name, float factor=1.)

  write the current depth map into a file
- SLSTEREO_EXPORT_DLL bool writePointCloudAs (sl::zed::Camera ∗zed, sl::POINT_CLOUD_FORMAT format, std::string name, bool withColor=false, bool keepOccludedPoint=false)

  write the current point cloud into a file

Typedef Documentation

typedef struct sl::double3Struct double3

typedef struct sl::float3Struct float3

typedef unsigned char uchar

typedef struct sl::uchar3Struct uchar3

Function Documentation

SLSTEREO_EXPORT_DLL bool sl::writeDepthAs (sl::zed::Camera ∗ *zed*, sl::DEPTH_FORMAT *format*, std::string *name*, float *factor = 1 .*)

write the current depth map into a file

Parameters

DEPTH_FORMAT	: for file format
name	: the name (path) in which the depth will be saved
factor	: only for PNG and PGM, apply a gain to the depth value (default 1.) The maximum value is 65536, so you can set the Camera::setDepthClampValue to 20000 and give a factor to 3, Do not forget to scale (by 1./factor) the pixel value to get the real depth. The occlusions are represented by 0.

Return values

return	false if something wrong happen, else return true

Note

factor : only for PNG and PGM

SLSTEREO_EXPORT_DLL bool sl::writePointCloudAs (sl::zed::Camera * *zed*, sl::POINT_CLOUD_FORMAT *format*, std::string *name*, bool *withColor* = `false`, bool *keepOccludedPoint* = `false`)

write the current point cloud into a file

Parameters

POINT_CLOUD-_FORMAT	: for file format
name	: the name (path) in which the point cloud will be saved
withColor	: indicates if the color must be saved (default false). Not avaliable for XYZ and VTK
keepOccluded-Point	: indicates if the non avaliable data should be saved and set to 0 (default false), if set to true this give a Point Cloud with a size = height $*$ width

Return values

return	false if something wrong happen, else return true

Note

The color is not saved for XYZ and VTK files

sl::zed Namespace Reference

Classes

- class Camera

  The main class to use the ZED camera.
- class Mat

  This class is used to store a buffer (the pointer) of an image and the associated metadata (size, type...).
- struct resolution

  Width and Height of each left and right image.
- struct CamParameters

  Intrinsic parameters of one camera.
- struct StereoParameters

  Intrinsic parameters of each cameras, baseline and convergence (radians)
- struct InitParams

  parameters for ZED initialization. a default constructor is enable.

Enumerations

- enum DATA_TYPE { FLOAT, UCHAR }
- enum MAT_TYPE { CPU, GPU }
- enum MODE {
  NONE, PERFORMANCE, MEDIUM, QUALITY,
  LAST_MODE }

  Enumerate for the pre-defined disparity map computation parameters.
- enum VIEW_MODE {
  STEREO_LEFT, STEREO_RIGHT, STEREO_ANAGLYPH, STEREO_DIFF,
  STEREO_SBS, STEREO_OVERLAY, STEREO_LEFT_GREY, STEREO_RIGHT_GREY,
  LAST_VIEW_MODE }

  Enumerate for available output views for monitoring.
- enum SIDE {
  LEFT, RIGHT, LEFT_GREY, RIGHT_GREY,
  LAST_SIDE }

  Enumerate for the Left and Right side of stereo Camera.
- enum SENSING_MODE { FILL, STANDARD, LAST_SENSING_MODE }

Enumerate for the different types of disparity map computation.

- enum MEASURE {
  DISPARITY, DEPTH, CONFIDENCE, XYZ,
  XYZRGBA, LAST_MEASURE }

  Enumerate for the retrievable measure (each measure should be normalized to be displayed)
- enum ERRCODE {
  SUCCESS, NO_GPU_COMPATIBLE, NOT_ENOUGH_GPUMEM, ZED_NOT_AVAILABLE,
  ZED_RESOLUTION_INVALID, ZED_SETTINGS_FILE_NOT_AVAILABLE, INVALID_SVO_FILE, RECORDER_ERROR,
  INVALID_COORDINATE_SYSTEM, ZED_WRONG_FIRMWARE, LAST_ERRCODE }

  Enumerate for error code during the sl::zed::Camera::init.
- enum ZED_SELF_CALIBRATION_STATUS { SELF_CALIBRATION_NOT_CALLED, SELF_CALIBRATION_RUNNING, SELF_CALIBRATION_FAILED, SELF_CALIBRATION_SUCCESS }

  Status for self calibration. Since v0.9.3, self-calibration is done in background and start in the Init or Reset function.
- enum ZEDResolution_mode {
  HD2K, HD1080, HD720, VGA,
  LAST_RESOLUTION }

  Enumerate for available resolutions for ZED camera.
- enum ZEDCamera_settings {
  ZED_BRIGHTNESS, ZED_CONTRAST, ZED_HUE, ZED_SATURATION,
  ZED_GAIN, ZED_EXPOSURE, LAST_SETTINGS }

  Enumerate the available camera settings for the ZED camera (contrast, hue, saturation, gain...)
- enum TRACKING_STATE {
  TRACKING_RELOCDB_INIT, TRACKING_INIT, TRACKING_RELOC, TRACKING_LOST,
  TRACKING_GOOD, TRACKING_OFF, TRACKING_LAST }

  Enumerate the current state of the tracking.
- enum TRACKING_FRAME_STATE { TRACKING_FRAME_NORMAL, TRACKING_FRAME_KEYFRAME, TRACKING_FRAME_CLOSE, TRACKING_FRAME_LAST }

  Enumerate the current state of the tracked frame.
- enum MAT_TRACKING_TYPE { PATH, POSE }

  Define which type of position matrix is used to store the tracking position.
- enum UNIT {
  MILLIMETER, METER, INCH, FOOT,
  LAST_UNIT }

  Enumerate for available metric unit of the depth.
- enum COORDINATE_SYSTEM {
  IMAGE = 1, LEFT_HANDED = 2, RIGHT_HANDED = 4, ROBOTIC = 8,
  APPLY_PATH = 16, LAST_COORDINATE_SYSTEM = 32 }

  Enumerate for available coordinates systems available for the points cloud.

Functions

- cv::Mat slMat2cvMat (sl::zed::Mat slMat)

  The function cast a sl::zed::Mat into an openCV cv::Mat.
- sl::zed::Mat cvMat2slMat (cv::Mat &cvMat)

  The function cast an openCV cv::Mat into a sl::zed::Mat.
- COORDINATE_SYSTEM operator| (COORDINATE_SYSTEM a, COORDINATE_SYSTEM b)
- static std::string tracking_state2str (TRACKING_STATE state)
- static std::string errcode2str (ERRCODE err)
- static std::string statuscode2str (ZED_SELF_CALIBRATION_STATUS stat)
- static std::string sensing_mode2str (SENSING_MODE mode)
- static std::string mode2str (MODE mode)
- static MODE str2mode (std::string mode)
- static std::string unit2str (UNIT unit)
- static UNIT str2unit (std::string unit)

Variables

- static std::vector< resolution > zedResolution

Enumeration Type Documentation

enum DATA_TYPE

Enumerate defines the type of the stored elements

Enumerator

FLOAT float elements (require to cast the pointer)

UCHAR unsigned char 8 bits elements

enum MAT_TYPE

Enumerate defines the buffer type

Enumerator

CPU the buffer is stored in the memory (CPU memory)

GPU the buffer is stored in the CUDA device (Npp pointer, GPU memory)

Function Documentation

sl::zed::Mat sl::zed::cvMat2slMat (cv::Mat & *cvMat*) `[inline]`

The function cast an openCV cv::Mat into a sl::zed::Mat.

Return values

Output	image as a sl::zed::Mat

References Mat::setUp().

static std::string sl::zed::errcode2str (ERRCODE *err*) `[inline]`, `[static]`

References INVALID_SVO_FILE, NO_GPU_COMPATIBLE, NOT_ENOUGH_GPUMEM, RECORDER_ERROR, SUCCESS, ZED_NOT_AVAILABLE, ZED_RESOLUTION_INVALID, ZED_SETTINGS_FILE_NOT_AVAILABLE, and ZED_WRONG_FIRMWARE.

static std::string sl::zed::mode2str (MODE *mode*) `[inline]`, `[static]`

References MEDIUM, NONE, PERFORMANCE, and QUALITY.

COORDINATE_SYSTEM sl::zed::operator| (COORDINATE_SYSTEM *a*, COORDINATE_SYSTEM *b*) `[inline]`

static std::string sl::zed::sensing_mode2str (SENSING_MODE *mode*) `[inline]`, `[static]`

References FILL, and STANDARD.

cv::Mat sl::zed::slMat2cvMat (sl::zed::Mat *slMat*) `[inline]`

The function cast a sl::zed::Mat into an openCV cv::Mat.

Return values

Output	image as a cv::Mat (opencv format)

References Mat::channels, Mat::data, Mat::data_type, Mat::getDataSize(), Mat::height, and Mat::width.

static std::string sl::zed::statuscode2str (ZED_SELF_CALIBRATION_STATUS *stat*) [inline], [static]

References SELF_CALIBRATION_FAILED, SELF_CALIBRATION_NOT_CALLED, SELF_CALIBRATION_RUNNING, and SELF_CALIBRATION_SUCCESS.

static MODE sl::zed::str2mode (std::string *mode*) [inline], [static]

References MEDIUM, NONE, PERFORMANCE, and QUALITY.

static UNIT sl::zed::str2unit (std::string *unit*) [inline], [static]

References FOOT, INCH, METER, and MILLIMETER.

static std::string sl::zed::tracking_state2str (TRACKING_STATE *state*) [inline], [static]

References TRACKING_GOOD, TRACKING_INIT, TRACKING_LOST, TRACKING_OFF, TRACKING_RELOC, and TRACKING_RELOCDB_INIT.

static std::string sl::zed::unit2str (UNIT *unit*) [inline], [static]

Variable Documentation

std::vector<resolution> zedResolution [static]

Initial value:

```
= [] {
            std::vector<resolution> v;
            v.push_back(resolution(2208, 1242));
            v.push_back(resolution(1920, 1080));
            v.push_back(resolution(1280, 720));
            v.push_back(resolution(672, 376));
            return v;
        }()
```

Available mode for the ZED camera sensors

Class Documentation

Camera Class Reference

The main class to use the ZED camera.

Public Member Functions

- Camera (ZEDResolution_mode zedRes_mode=ZEDResolution_mode::HD720, float fps=0.0, int zed_linux_id=0)

Camera constructor. The ZEDResolution_mode sets the sensor resolution and defines the size of the output images, including the measures (disparity map, confidence map..).

- Camera (std::string zed_record_filename)

  Camera constructor, from svo file previously recorded. The size of the output is defined by the recorded images.
- ~Camera ()
- ERRCODE init (InitParams &parameters)

  The init function must be called after the instantiation. The function checks if the ZED camera is plugged and opens it, if the graphics card is compatible, allocates the memory and launches the automatic calibration.
- bool grab (SENSING_MODE dm_type=SENSING_MODE::STANDARD, bool computeMeasure=1, bool computeDisparity=1, bool computeXYZ=1)

  The function grabs a new image, rectifies it and computes the disparity map and optionally the depth map. The grabbing function is typically called in the main loop.
- Mat getView (VIEW_MODE view)

  Gets a CPU image to display. Several modes available SidebySide, anaglyph... (for more see sl::zed::VIEW_MODE)
- Mat getView_gpu (VIEW_MODE view)

  Gets a GPU image to display. Several modes available SidebySide, anaglyph... (for more see sl::zed::VIEW_MODE)
- Mat retrieveImage (SIDE side)

  Downloads the rectified image from the device and returns the CPU buffer. The retrieve function should be called after the function Camera::grab.
- Mat retrieveMeasure (MEASURE measure)

  Downloads the measure (disparity, depth or confidence of disparity) from the device and returns the CPU buffer. The retrieve function should be called after the function Camera::grab.
- Mat retrieveImage_gpu (SIDE side)

  Gets the rectified image GPU buffer. The retrieve function should be called after the function Camera::grab.
- Mat retrieveMeasure_gpu (MEASURE measure)

  Gets the measure (disparity, depth or certainty of disparity) GPU buffer. The retrieve function should be called after the function Camera::grab.
- Mat normalizeMeasure (MEASURE measure, float min=0, float max=0)

  Performs a GPU normalization of the measure value and download the result as a CPU image.
- Mat normalizeMeasure_gpu (MEASURE measure, float min=0, float max=0)

  GPU Normalization of the measure value and get the result as a GPU image.
- void setConfidenceThreshold (int ThresholdIdx)

  Sets a filtering value for the disparity map (and by extension the depth map). The function should be called before Camera::grab to be taken into account.
- int getConfidenceThreshold ()

  Gets the current confidence threshold value for the disparity map (and by extension the depth map).
- CUcontext getCUDAContext ()

  Gets the CUDA context used for all the computation.
- resolution getImageSize ()

  Gets the image size.
- void setDepthClampValue (int distanceMax)

  Set the maximum distance of depth/disparity estimation (all values after this limit will be reported as TOO_FAR value)
- float getDepthClampValue ()

  Get the current maximum distance of depth/disparity estimation.
- float getClosestDepthValue ()

  Get the closest measurable distance by the camera, according to the camera and the depth map parameters.

Camera infos

- unsigned int getZEDSerial ()

  Gets the ZED Serial Number.
- unsigned int getZEDFirmware ()

Gets the ZED Current Firmware version.

- bool setSVOPosition (int frame)

  Sets the position of the SVO file to a desired frame.
- int getSVOPosition ()

  Get the current position of the SVO file.
- int getSVONumberOfFrames ()

  Get the number of frames in the SVO file.
- void setCameraSettingsValue (ZEDCamera_settings mode, int value, bool usedefault=false)

  Set the value to the corresponding Camera Settings mode (Gain, brightness, hue, exposure...)
- int getCameraSettingsValue (ZEDCamera_settings mode)

  Get the current value to the corresponding Camera Settings mode (Gain, brightness, hue, exposure...)
- float getCurrentFPS ()

  Get the current fps of the camera.
- bool setFPS (int desiredFPS)

  Set a new frame rate for the camera, or the closest avaliable frame rate.
- long long getCameraTimestamp ()

  Get the timestamp at the time the frame has been extracted from USB stream. (should be called after a grab())
- long long getCurrentTimestamp ()

  Get the current timestamp at the time the function is called. Can be compared to the camera timestamp for synchronization.
- void setFlip (bool flip)

  Useful if you use the camera upside down, this will flip the images so you can get the images in a normal way.

Self calibration

- StereoParameters * getParameters ()

  Gets the ZED parameters.
- ZED_SELF_CALIBRATION_STATUS getSelfCalibrationStatus ()

  Get the current status of the self-calibration.
- sl::double3 getSelfCalibrationRotation ()

  Get the estimated rotation from the self-calibration.
- bool resetSelfCalibration ()

  The reset function can be called at any time AFTER the Camera::init function has been called. It will reset and calculate again correction for misalignment, convergence and color mismatch. It can be called after changing camera parameters without needing to restart your executable.

Tracking

- bool enableTracking (Eigen::Matrix4f &initPosition, bool enableAreaLearning=false, std::string areaDBpath="")

  Initialize and Start the tracking functions.
- sl::zed::TRACKING_STATE getPosition (Eigen::Matrix4f &position, MAT_TRACKING_TYPE mat-_type=MAT_TRACKING_TYPE::PATH)

  fill the position of the camera and return the current state of the Tracker
- int getTrackingConfidence ()

  Return a confidence metric of the tracking [0-100], 0 means that the tracking is lost.
- TRACKING_FRAME_STATE getTrackingFrameState ()

  return the state of the current tracked frame
- bool saveAreaLearningDB (std::string areaDBpath)

  save the area learning information in a file.
- void stopTracking ()

  stop the tracker, if you want to restart, call enableTracking().
- bool setTrackingPrior (Eigen::Matrix4f &priorRt)

  if you have an external sensor (IMU) you can use its data to set a prior for the tracking.
- bool setTrackingPrior (Eigen::Matrix3f &priorR)

  if you have an external sensor you can use its data to set a prior for the tracking.
- bool setTrackingPrior (Eigen::Vector3f &priorT)

  if you have an external sensor you can use its data to set a prior for the tracking.

Recorder

- ERRCODE initRecording (std::string filepath, bool avi_file=false, bool side_by_side=true)
  Initializes the recorder.
- bool record ()
  Records one camera frame. Typically called in a loop, without calling grab function.
- sl::zed::Mat getCurrentRawRecordedFrame ()
  Get the current side by side YUV 4:2:2 frame, CPU buffer.
- void stopRecording ()
  Stops the recording and closes the file.
- void displayRecorded ()
  Displays the current recorded frame.

Static Public Member Functions

- static std::string getSDKVersion ()
  The function return the version of the currently installed ZED SDK.
- static int isZEDconnected ()
  The function checks if ZED cameras are connected, can be called before instantiating a Camera object.
- static int sticktoCPUCore (int cpu_core)
  The function stick the calling thread to a specific CPU core. This function is only available for Jetson TK1 and TX1.

Detailed Description

The main class to use the ZED camera.

Constructor & Destructor Documentation

Camera (ZEDResolution_mode *zedRes_mode* = `ZEDResolution_mode::HD720`, float *fps* = `0.0`, int *zed_linux_id* = `0`)

Camera constructor. The ZEDResolution_mode sets the sensor resolution and defines the size of the output images, including the measures (disparity map, confidence map..).

All computation is done on a CUDA capable device (That means that every CPU computation will need a memory retrieve of the images, which takes some time). If the performance is the main focus, all external computation should run on GPU. The retrieve∗_gpu gives directly access to the gpu buffer.

Parameters

zedRes_mode	: the chosen ZED resolution
fps	: a requested fps for this resolution. set as 0.0 will choose the default fps for this resolution (see User guide)
zed_linux_id	: ONLY for LINUX : if multiple ZEDs are connected, it will choose the first zed listed (if zed_linux_id=0), the second listed (if zed_linux_id=1), ... Each ZED will create its own memory (CPU and GPU), therefore the number of ZED available will depend on the configuration of your computer. Currently not available for Windows

Camera (std::string *zed_record_filename*)

Camera constructor, from svo file previously recorded. The size of the output is defined by the recorded images.

Parameters

zed_record_filename	: path with filename to the recorded svo file

~Camera ( )

Member Function Documentation

void displayRecorded ( )

Displays the current recorded frame.

Warning

Might reduce the recording framerate

bool enableTracking (Eigen::Matrix4f & *initPosition*, bool *enableAreaLearning* = `false`, std::string *areaDBpath* = `""`)

Initialize and Start the tracking functions.

Parameters

initPosition	: position of the first camera, used as reference. By default it should be identity.
enableAreaLearning	: define if the relocalization (when tracking is lost) and loop closure is enable or not.
areaDBpath	: define if and where a relocalization database from a previous run on the same scene has to be loaded.

Return values

bool	: return false if the area database file wasn't found, true otherwise

int getCameraSettingsValue (ZEDCamera_settings *mode*)

Get the current value to the corresponding Camera Settings mode (Gain, brightness, hue, exposure...)

Parameters

ZEDCamera_settings	: enum for the control mode

Return values

Current	value for the corresponding control (-1 if something wrong happened).

long long getCameraTimestamp ( )

Get the timestamp at the time the frame has been extracted from USB stream. (should be called after a grab())

Return values

Current	Timestamp in ns. Return -1 if working with svo files.

float getClosestDepthValue ( )

Get the closest measurable distance by the camera, according to the camera and the depth map parameters.

Return values

float	: the distance in the defined UNIT

int getConfidenceThreshold ( )

Gets the current confidence threshold value for the disparity map (and by extension the depth map).

Return values

	current filtering value between 0 and 100.

CUcontext getCUDAContext ( )

Gets the CUDA context used for all the computation.

Return values

CUDA_context	: the CUcontext

float getCurrentFPS ( )

Get the current fps of the camera.

Return values

Current	FPS. Return 0 if working with SVO files or -1.f if something goes wrong.

sl::zed::Mat getCurrentRawRecordedFrame ( )

Get the current side by side YUV 4:2:2 frame, CPU buffer.

Warning

> The buffer must be duplicated if record() is called, it will be overwritten otherwise. The buffer must not be freed. Undefined behavior if called outside recorder mode.

long long getCurrentTimestamp ( )

Get the current timestamp at the time the function is called. Can be compared to the camera timestamp for synchronization.

Return values

Current	Timestamp in ns. Return -1 if working with svo files.

float getDepthClampValue ( )

Get the current maximum distance of depth/disparity estimation.

Return values

	current maximum distance in the defined UNIT

resolution getImageSize ( )

Gets the image size.

Return values

resolution	: the image size

StereoParameters* getParameters ( )

Gets the ZED parameters.

Return values

StereoParameters	pointer containing the intrinsic parameters of each camera and the baseline (mm) and convergence (radian) of the ZED.

sl::zed::TRACKING_STATE getPosition (Eigen::Matrix4f & *position*, MAT_TRACKING_TYPE *mat_type* = `MAT_TRACKING_TYPE::PATH`)

fill the position of the camera and return the current state of the Tracker

Parameters

position	: the matrix containing the position of the camera (path or position) 4x4 Matrix position/path : $\begin{pmatrix} R11 & R21 & R31 & Tx \\ R12 & R22 & R32 & Ty \\ R13 & R23 & R33 & Tz \\ 0 & 0 & 0 & 1 \end{pmatrix}$

-> *Rnm* is the XYZ 3x3 rotation matrix

-> *Tx/y/z* is the XYZ translation vector.

Extract Rotation Matrix -> Eigen::Matrix3f rot_ = position.block(0,0,3,3);

Extract Translation Vector -> Eigen::Vector3f trans_ = position.block(0,3,3,1);

Convert to Quaternion -> Eigen::Quaternionf quat(position.block(0,0,3,3));

Parameters

mat_type	: define if the function return the path (the cumulate displacement of the camera) or juste the pose (the displacement from the previous position).

static std::string getSDKVersion ( ) `[static]`

The function return the version of the currently installed ZED SDK.

Return values

ZED	SDK version as a string with the following format : MAJOR.MINOR.PATCH

sl::double3 getSelfCalibrationRotation ( )

Get the estimated rotation from the self-calibration.

Return values

double3	(structure of 3 double values) of the Rotation (one component for each axis) - d1 will return the Rx component ("Tilt" or "Pitch") - d2 will return the Ry component ("Convergence") - d3 will return the Rz component ("Roll") All values are in radians.

ZED_SELF_CALIBRATION_STATUS getSelfCalibrationStatus ( )

Get the current status of the self-calibration.

Return values

ZED_SELF_CALIBRATION_STATUS	: The status code gives information about the self calibration status. For more details see sl::zed::ZED_SELF_CALIBRATION_STATUS.

int getSVONumberOfFrames ( )

Get the number of frames in the SVO file.

Return values

SVO	Style Only : the total number of frames in the SVO file (-1 if the SDK is not reading a SVO)

int getSVOPosition ( )

Get the current position of the SVO file.

Return values

SVO	Style Only : the current position in the SVO file as int (-1 if the SDK is not reading a SVO)

int getTrackingConfidence ( )

Return a confidence metric of the tracking [0-100], 0 means that the tracking is lost.

TRACKING_FRAME_STATE getTrackingFrameState ( )

return the state of the current tracked frame

Mat getView (VIEW_MODE *view*)

Gets a CPU image to display. Several modes available SidebySide, anaglyph... (for more see sl::zed::VIEW_MODE)

Parameters

view	: the wanted mode

Return values

View	: the return value is a CPU sl::zed::Mat, 4 channels, UCHAR.

Warning

The buffer must be duplicated if another CPU retrieve has to be done, it will be overwritten otherwise. The buffer must not be freed.

Mat getView_gpu (VIEW_MODE *view*)

Gets a GPU image to display. Several modes available SidebySide, anaglyph... (for more see sl::zed::VIEW_MODE)

Parameters

view	: the wanted mode

Return values

View	: the return value is a GPU sl::zed::Mat, 4 channels, UCHAR.

Warning

The buffer must be duplicated if another GPU retrieve has to be done, it will be overwritten otherwise. The buffer must not be freed.

unsigned int getZEDFirmware ( )

Gets the ZED Current Firmware version.

Return values

Returns	the ZED Firmware version (as uint), 0 if the ZED is not connected.

unsigned int getZEDSerial ( )

Gets the ZED Serial Number.

Return values

Returns	the ZED Serial Number (as uint) (Live or SVO).

bool grab (SENSING_MODE *dm_type* = `SENSING_MODE::STANDARD`, bool *computeMeasure* = `1`, bool *computeDisparity* = `1`, bool *computeXYZ* = `1`)

The function grabs a new image, rectifies it and computes the disparity map and optionally the depth map. The grabbing function is typically called in the main loop.

Parameters

dm_type	: defines the type of disparity map, more info : sl::zed::SENSING_MODE definition
computeMeasure	: defines if the depth map should be computed. If false, only the disparity map is computed.
computeDisparity	: defines if the disparity map should be computed. If false, the depth map can't be computed and only the camera images can be retrieved.
computeXYZ	: defines if the point cloud should be computed (including XYZRGBA).

Return values

error	: the function returns false if no problem was encountered, true otherwise.

Precondition

Camera::init function must have been called (once for the lifetime of the Camera object) before calling Camera::grab.

ERRCODE init (InitParams & *parameters*)

The init function must be called after the instantiation. The function checks if the ZED camera is plugged and opens it, if the graphics card is compatible, allocates the memory and launches the automatic calibration.

Parameters

parameters	: a structure containing all the individual parameters

Return values

ERRCODE	: The error code gives information about the internal process, if SUCCESS is returned, the camera is ready to use. Every other code indicates an error and the program should be stopped. For more details see sl::zed::ERRCODE.

ERRCODE initRecording (std::string *filepath*, bool *avi_file* = `false`, bool *side_by_side* = `true`)

Initializes the recorder.

Parameters

filepath	: path with absolute or relative filename of the recorded file
avi_file	: record in avi (not compatible with the ZED SDK as an input) rather than svo
side_by_side	: record one avi file with side by side images, left and right otherwise

Note

The disparity/depth map is neither recorded nor computed

Warning

Camera::init mustn't be called.

static int isZEDconnected ( ) `[static]`

The function checks if ZED cameras are connected, can be called before instantiating a Camera object.

Return values

the	number of connected ZED

Warning

On Windows, only one ZED is accessible so this function will return 1 even if multiple ZED are connected.

Mat normalizeMeasure (MEASURE *measure*, float *min* = *0*, float *max* = *0*)

Performs a GPU normalization of the measure value and download the result as a CPU image.

Parameters

MEASURE	: defines the requested output (see sl::zed::MEASURE)
min	: defines the lower bound of the normalization, default : automatically found
max	: defines the upper bound of the normalization, default : automatically found

Return values

normalized_measure	: the return value is a CPU sl::zed::Mat, 4 channels, UCHAR.

Warning

The buffer must be duplicated if another CPU retrieve has to be done, it'll be overwritten otherwise. The buffer must not be freed.

Mat normalizeMeasure_gpu (MEASURE *measure*, float *min = 0*, float *max = 0*)

GPU Normalization of the measure value and get the result as a GPU image.

Parameters

MEASURE	: defines the requested output (see sl::zed::MEASURE)
min	: defines the lower bound of the normalization, default : automatically found
max	: defines the upper bound of the normalization, default : automatically found

Return values

normalized_measure	: the return value is a GPU sl::zed::Mat, 4 channels, UCHAR.

Warning

The buffer must be duplicated if another GPU retrieve has to be done, it'll be overwritten otherwise. The buffer must not be freed.

bool record ( )

Records one camera frame. Typically called in a loop, without calling grab function.

Warning

Camera::grab mustn't be called.

bool resetSelfCalibration ( )

The reset function can be called at any time AFTER the Camera::init function has been called. It will reset and calculate again correction for misalignment, convergence and color mismatch. It can be called after changing camera parameters without needing to restart your executable.

Return values

error	boolean value : the function returns false if no problem was encountered, true otherwise. if no problem was encountered, the camera will use new parameters. Otherwise, it will be the old ones

Mat retrieveImage (SIDE *side*)

Downloads the rectified image from the device and returns the CPU buffer. The retrieve function should be called after the function Camera::grab.

Parameters

SIDE	: defines the image side wanted (see sl::zed::SIDE)

Return values

Image	: the return value is a CPU sl::zed::Mat, 4 channels, UCHAR. The size is given by the input parameters of the constructor Camera.

Warning

The buffer must be duplicated if another CPU retrieve has to be done, it will be overwritten otherwise. The buffer must not be freed.

Mat retrieveImage_gpu (SIDE *side*)

Gets the rectified image GPU buffer. The retrieve function should be called after the function Camera::grab.

Parameters

SIDE	: defines the image side wanted (see sl::zed::SIDE)

Return values

Image	: the return value is a GPU sl::zed::Mat, 4 channels, UCHAR. The size is given by the input parameters of the constructor Camera.

Warning

The buffer must not be freed.

Mat retrieveMeasure (MEASURE *measure*)

Downloads the measure (disparity, depth or confidence of disparity) from the device and returns the CPU buffer. The retrieve function should be called after the function Camera::grab.

Parameters

MEASURE	: defines the type wanted, such as disparity map, depth map or the confidence (see sl::zed::MEASURE)

Return values

Measure	: the return value is a CPU sl::zed::Mat, 1 or 4 channels, FLOAT

Warning

The buffer must be duplicated if another CPU retrieve has to be done, it will be overwritten otherwise. The buffer must not be freed.

Mat retrieveMeasure_gpu (MEASURE *measure*)

Gets the measure (disparity, depth or certainty of disparity) GPU buffer. The retrieve function should be called after the function Camera::grab.

Parameters

MEASURE	: defines the data type wanted (see sl::zed::MEASURE)

Return values

Measure	: the return value is a GPU sl::zed::Mat, 1 or 4 channels, FLOAT. The size is given by the input parameters of the constructor Camera.

Warning

The buffer must not be freed.

bool saveAreaLearningDB (std::string *areaDBpath*)

save the area learning information in a file.

Parameters

areaDBpath	: the name and path of the database

void setCameraSettingsValue (ZEDCamera_settings *mode*, int *value*, bool *usedefault* = `false`)

Set the value to the corresponding Camera Settings mode (Gain, brightness, hue, exposure...)

Parameters

ZEDCamera_settings	: enum for the control mode
value	: value to set for the corresponding control
usedefault	: will set default (or automatic) value if set to true (value (int) will not be taken into account)

void setConfidenceThreshold (int *ThresholdIdx*)

Sets a filtering value for the disparity map (and by extension the depth map). The function should be called before Camera::grab to be taken into account.

Parameters

ThresholdIdx	: a value in [1,100]. A lower value means more confidence and precision (but less density), an upper value reduces the filtering (more density, less certainty). Other value means no filtering.

void setDepthClampValue (int *distanceMax*)

Set the maximum distance of depth/disparity estimation (all values after this limit will be reported as TOO_FAR value)

Parameters

distanceMax	: maximum distanec in the defined UNIT

void setFlip (bool *flip*)

Useful if you use the camera upside down, this will flip the images so you can get the images in a normal way.

Parameters

flip	: apply or not the vertical flip

bool setFPS (int *desiredFPS*)

Set a new frame rate for the camera, or the closest avaliable frame rate.

Parameters

desiredFPS	: the new desired frame rate

Return values

bool	: return an error if something wrong happen

bool setSVOPosition (int *frame*)

Sets the position of the SVO file to a desired frame.

Parameters

frame	: the number of the desired frame to be decoded.

Return values

true	if succes, false if failed (i.e. if the ZED is currently used in live and not playing a SVO file)

bool setTrackingPrior (Eigen::Matrix4f & *priorRt*)

if you have an external sensor (IMU) you can use its data to set a prior for the tracking.

Parameters

prior	: the pose (rotations and translations) that will be use as prior for tracking operation.

Return values

false	if the tracking is not enabled, 1 otherwise.

bool setTrackingPrior (Eigen::Matrix3f & *priorR*)

if you have an external sensor you can use its data to set a prior for the tracking.

Parameters

prior	: the pose (rotations only) that will be use as prior for tracking operation.

Return values

false	if the tracking is not enabled, 1 otherwise.

bool setTrackingPrior (Eigen::Vector3f & *priorT*)

if you have an external sensor you can use its data to set a prior for the tracking.

Parameters

prior	: the pose (translations only) that will be use as prior for tracking operation.

Return values

false	if the tracking is not enabled, 1 otherwise.

static int sticktoCPUCore (int *cpu_core*) [static]

The function stick the calling thread to a specific CPU core. This function is only available for Jetson TK1 and TX1.

Parameters

cpu_core	: int that defines the core the thread must be run on. could be between 0 and 3. (cpu0,cpu1,cpu2,cpu3).

Return values

0	is stick is OK, otherwise status error.

Warning

Function only available for Jetson. On other platform, result will be always 0 and no operations are performed.

void stopRecording ( )

Stops the recording and closes the file.

void stopTracking ( )

stop the tracker, if you want to restart, call enableTracking().

CamParameters Struct Reference

Intrinsic parameters of one camera.

Public Member Functions

- void SetUp (float fx_, float fy_, float cx_, float cy_)

Public Attributes

- float fx
- float fy
- float cx
- float cy
- double disto [5]
- float vFOV
- float hFOV
- float dFOV

Detailed Description

Intrinsic parameters of one camera.

Member Function Documentation

void SetUp (float *fx_*, float *fy_*, float *cx_*, float *cy_*) `[inline]`

References CamParameters::cx, CamParameters::cy, CamParameters::fx, and CamParameters::fy.

Member Data Documentation

float cx

Optical center x

Referenced by CamParameters::SetUp().

float cy

Optical center y

Referenced by CamParameters::SetUp().

float dFOV

Diagonal field of view after stereo rectification

double disto[5]

Distortion factor : k1, k2, k3, r1, r2

float fx

Focal x

Referenced by CamParameters::SetUp().

float fy

Focal y

Referenced by CamParameters::SetUp().

float hFOV

Horizontal field of view after stereo rectification

float vFOV

Vertical field of view after stereo rectification

double3Struct Struct Reference

Public Member Functions

- double3Struct (float d1_=0, float d2_=0, float d3_=0)

Public Attributes

- double d1
- double d2
- double d3

Constructor & Destructor Documentation

double3Struct (float *d1_ = 0*, float *d2_ = 0*, float *d3_ = 0*) [inline]

References double3Struct::d1, double3Struct::d2, and double3Struct::d3.

Member Data Documentation

double d1

Referenced by double3Struct::double3Struct().

double d2

Referenced by double3Struct::double3Struct().

double d3

Referenced by double3Struct::double3Struct().

float3Struct Struct Reference

Public Member Functions

- float3Struct (float f1_=0, float f2_=0, float f3_=0)

Public Attributes

- float f1
- float f2
- float f3

Constructor & Destructor Documentation

float3Struct (float *f1_ = 0*, float *f2_ = 0*, float *f3_ = 0*) [inline]

References float3Struct::f1, float3Struct::f2, and float3Struct::f3.

Member Data Documentation

float f1

Referenced by float3Struct::float3Struct().

float f2

Referenced by float3Struct::float3Struct().

float f3

Referenced by float3Struct::float3Struct().

InitParams Struct Reference

parameters for ZED initialization. a default constructor is enable.

Public Member Functions

- InitParams (MODE mode_=PERFORMANCE, UNIT unit_=MILLIMETER, COORDINATE_SYSTEM coord_=IMAGE, bool verbose_=false, int device_=-1, float minDist_=-1., bool disable=false, bool vflip_=false)
- void save (std::string filename)

  Save the current bunch of parameters into a file.
- bool load (std::string filename)

  load the values of the parameters contained in a file.

Public Attributes

- MODE mode
- UNIT unit
- COORDINATE_SYSTEM coordinate
- bool verbose
- int device
- float minimumDistance
- bool disableSelfCalib
- bool vflip

Detailed Description

parameters for ZED initialization. a default constructor is enable.

Constructor & Destructor Documentation

InitParams (MODE *mode_* = PERFORMANCE, UNIT *unit_* = MILLIMETER, COORDINATE_SYSTEM *coord_* = IMAGE, bool *verbose_* = `false`, int *device_* = `-1`, float *minDist_* = `-1.`, bool *disable* = `false`, bool *vflip_* = `false`) `[inline]`

Member Function Documentation

bool load (std::string *filename*)

load the values of the parameters contained in a file.

Parameters

filename	: the name of the file.

Returns

Returns if the file was succefully loaded or not.

void save (std::string *filename*)

Save the current bunch of parameters into a file.

Parameters

filename	: the name of the file in which the parameters will be stored.

Member Data Documentation

COORDINATE_SYSTEM coordinate

define the coordinate system, in which the path and the point cloud is referred. default COORDINATE_SYSTEM::IMAGE

int device

defines the graphics card on which the computation will be done. The default value -1 search the more powerful usable GPU. default -1

bool disableSelfCalib

if set to true, it will disable self-calibration and take the optional calibration parameters without optimizing them. It is advised to leave it as false, so that calibration parameters can be optimized. default false

float minimumDistance

specify the minimum depth information that will be computed, in the unit you previously define.

Note

By default we compute disparities in a constant interval, defined by a ratio of the image size. e.g. for HD720 : $W = 1280, interval(cst) = \frac{1}{8} \rightarrow DisparityMax = \frac{1280}{8} = 160 \rightarrow DepthMin = \frac{focal*baseline}{DisparityMax} = \frac{700*120}{160} = 525mm$ default int -1

Warning

The computation time is affected by the value, exponentially. The closer it gets the longer the disparity search will take. In case of limited computation power, consider increasing the value.

MODE mode

defines the quality of the disparity map, affects the level of details and also the computation time. defult : MODE::PERFORMANCE

UNIT unit

define the unit metric for all the depth-distance values. default UNIT::MILLIMETER

bool verbose

if set to true, it will output some information about the current status of initialization. default false

bool vflip

if set to true, it will vertically flipped frames coming from the ZED. (for "Hang out" mode). default false

Mat Class Reference

This class is used to store a buffer (the pointer) of an image and the associated metadata (size, type...).

Public Member Functions

- Mat ()

  Mat default constructor.
- Mat (int width, int height, int channels, DATA_TYPE data_type, sl::uchar ∗data, MAT_TYPE mat_type=MAT_TYPE::CPU)

  Mat constructor.
- sl::uchar3 getValue (int x, int y) const

  Gets the value of the image at the given coordinate.
- void setUp (int width, int height, int step, sl::uchar ∗data, int channels, DATA_TYPE data_type, MAT_TYPE mat_type=CPU)

  Sets up the buffer metadata.
- void setUp (int width, int height, int channels, DATA_TYPE data_type, MAT_TYPE mat_type=CPU)

  Sets up the buffer metadata.
- void allocate_cpu (int width, int height, int channels, DATA_TYPE data_type)

  Allocates the memory.
- void allocate_cpu ()

  Allocates the memory with the metadata previously set.
- void deallocate ()

  Deallocates the memory.
- int getWidthByte () const

  Gets the width of the image in bytes, useful for the npp function (take into account the number of channels)
- int getDataSize () const

  Gets the size of the buffer type.
- void info ()

  Prints in the standard output information about the Mat.

Public Attributes

- int width
- int height
- int step
- sl::uchar ∗ data
- int channels
- DATA_TYPE data_type
- MAT_TYPE type

Detailed Description

This class is used to store a buffer (the pointer) of an image and the associated metadata (size, type...).

Constructor & Destructor Documentation

Mat ( )

Mat default constructor.

Mat (int *width*, int *height*, int *channels*, DATA_TYPE *data_type*, sl::uchar * *data*, MAT_TYPE *mat_type* = *MAT_TYPE::CPU*)

Mat constructor.

Parameters

width	: width of the image in pixels
height	: height of the image in pixels
channels	: number of channels in the image
data_type	: type of stored element (see sl::zed::DATA_TYPE). Can be float or uchar.
data	: the buffer
mat_type	: defines where the buffer is stored (CPU or GPU memory)

Warning

> The buffer memory is NEITHER automatically allocated NOR freed and requires an explicit call of the corresponding function (Exception : for some of the sl::zed::Camera functions, the sl::zed::Mat may already be pre-allocated and must not be freed. For more information, refer to the corresponding function documentation)

Member Function Documentation

void allocate_cpu (int *width*, int *height*, int *channels*, DATA_TYPE *data_type*)

Allocates the memory.

Parameters

width	: width of the image in pixels
height	: height of the image in pixels
channels	: number of channels in the image
data_type	: type of element stored

Warning

> The Mat is required to be a CPU Mat

void allocate_cpu ( )

Allocates the memory with the metadata previously set.

Precondition

> All the metadata must have been filled before calling allocate to prevent an undefined behavior

Warning

The Mat is required to be a CPU Mat

void deallocate ( )

Deallocates the memory.

int getDataSize ( ) const

Gets the size of the buffer type.

Note

sizeof() is called on Mat::data using the real type

Return values

size	in bytes

Referenced by sl::zed::slMat2cvMat().

sl::uchar3 getValue (int *x*, int *y*) const

Gets the value of the image at the given coordinate.

Parameters

x	coordinate
y	coordinate

Return values

pixel_value	If the Mat has less than 3 channels a uchar3 is still produced with the value 0 for the missing channels. If the Mat has more than 3 channels the value correspond to the first 3 channels and the others are ignored

int getWidthByte ( ) const

Gets the width of the image in bytes, useful for the npp function (take into account the number of channels)

Return values

width_byte	: the width in bytes

void info ( )

Prints in the standard output information about the Mat.

void setUp (int *width*, int *height*, int *step*, sl::uchar * *data*, int *channels*, DATA_TYPE *data_type*, MAT_TYPE *mat_type* = CPU)

Sets up the buffer metadata.

Parameters

width	: width of the image in pixels
height	: height of the image in pixels
step	: the step in bytes (essential if the data are aligned)
data	: the buffer
channels	: number of channels in the image
data_type	: type of element stored
mat_type	: define where the buffer is stored (CPU or GPU memory)

Referenced by sl::zed::cvMat2slMat().

void setUp (int *width*, int *height*, int *channels*, DATA_TYPE *data_type*, MAT_TYPE *mat_type* = CPU)

Sets up the buffer metadata.

Parameters

width	: width of the image in pixels
height	: height of the image in pixels
channels	: number of channels in the image
data_type	: type of element stored
mat_type	: defines where the buffer is stored (CPU or GPU memory)

Note

if the mat_type is CPU, the step is automatically set

Member Data Documentation

int channels

Referenced by sl::zed::slMat2cvMat().

sl::uchar* data

The pointer to the data, can also be casted to store float value

Referenced by sl::zed::slMat2cvMat().

DATA_TYPE data_type

Referenced by sl::zed::slMat2cvMat().

int height

Referenced by sl::zed::slMat2cvMat().

int step

MAT_TYPE type

int width

Referenced by sl::zed::slMat2cvMat().

resolution Struct Reference

Width and Height of each left and right image.

Public Member Functions

- resolution (int w_, int h_)

Public Attributes

- int width
- int height

Detailed Description

Width and Height of each left and right image.

Constructor & Destructor Documentation

resolution (int *w_*, int *h_*) `[inline]`

References resolution::height, and resolution::width.

Member Data Documentation

int height

Height of single image in pixels

Referenced by resolution::resolution().

int width

Width of single image in pixels

Referenced by resolution::resolution().

StereoParameters Struct Reference

Intrinsic parameters of each cameras, baseline and convergence (radians)

Public Attributes

- float baseline
- float Ty
- float Tz
- float convergence
- float Rx

- float Rz
- CamParameters LeftCam
- CamParameters RightCam

Detailed Description

Intrinsic parameters of each cameras, baseline and convergence (radians)

Member Data Documentation

float baseline

Distance between the 2 cameras in mm

float convergence

Convergence between the 2 cameras in radians

CamParameters LeftCam

Intrinsic parameters of the left camera

CamParameters RightCam

Intrinsic parameters of the right camera

float Rx

Rotation around X axis ("tilt"), will be affined by self-calibration - optional

float Rz

Rotation around Z axis("roll"), will be affined by self-calibration - optional

float Ty

float Tz

uchar3Struct Struct Reference

Public Member Functions

- uchar3Struct (uchar c1_=0, uchar c2_=0, uchar c3_=0)
- void setValue (uchar c1_, uchar c2_=0, uchar c3_=0)

Public Attributes

- uchar c1
- uchar c2
- uchar c3

Constructor & Destructor Documentation

uchar3Struct (uchar *c1_* = *0*, uchar *c2_* = *0*, uchar *c3_* = *0*) `[inline]`

References uchar3Struct::c1, uchar3Struct::c2, and uchar3Struct::c3.

Member Function Documentation

void setValue (uchar *c1_*, uchar *c2_* = *0*, uchar *c3_* = *0*) `[inline]`

References uchar3Struct::c1, uchar3Struct::c2, and uchar3Struct::c3.

Member Data Documentation

uchar c1

Referenced by uchar3Struct::setValue(), and uchar3Struct::uchar3Struct().

uchar c2

Referenced by uchar3Struct::setValue(), and uchar3Struct::uchar3Struct().

uchar c3

Referenced by uchar3Struct::setValue(), and uchar3Struct::uchar3Struct().

Index