Qualcomm Technologies, Inc.

Machine Vision

Application Programming Interface

80-H9220-2 Rev. B

January 15, 2018

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

1 Introduction

The release numbering scheme follows conventions in `www.semver.org`

1.1 Overview

Qualcomm's Machine Vision SDK provides highly runtime optimized and state of the art computer vision algorithms to enable such features as localization, autonomy, and obstacle avoidance. Some example features included are:

- Camera Auto Calibration (CAC) for online monocular camera calibration.
- Camera Parameter Adjustment (CPA) for auto gain and exposure control.
- Depth from Stereo (DFS) for dense depth mapping.
- Downward Facing Tracker (DFT) for relative localization.
- Stereo Auto-Calibration (SAC) for online calibration of a stereo camera rig.
- Sequence Reader/Write (SRW) for reading and writing MV data sequences.
- Visual Inertial Simultaneous Localization and Mapping (VISLAM) for 6-DOF localization and pose estimation.
- Voxel Map (VM) for 3D depth fusion and mapping.

2 Class Documentation

2.1 mvAttitudeData Struct Reference

```
#include <mvSRW.h>
```

2.1.1 Detailed Description

Attitude estimate.

Parameters:

timestamp	Timestamp of data in microseconds.

rotation_matrix	World to body rotation matrix (R) in row major order. Example:
	`a0 = [0 0 g]`
	`a = R^T * a0`
	`a = [-sin(pitch)`
	`cos(pitch) * sin(roll)`
	`cos(pitch) * cos(roll)] * g`
	where pitch, roll, and yaw are using Tait-Bryan ZYX convention and yaw from magnetic north.

2.2 mvCACConfiguration Struct Reference

```
#include <mvCAC.h>
```

2.2.1 Detailed Description

Configuration parameters for initializing mvCAC.

Parameters:

maxNumKeypoints	Maximum number of key points used for tracking.
tauGyroCamera	Initial value for time offset from camera to gyroscope, in microseconds.
tauRollingShutterSkew	Initial value for offset of rolling shutter skew, in microseconds.

principalPointErrorStddev	Standard deviation of the error between initial principal point and the ground truth, unit-less (normalized by image size).
focalRatioErrorStddev	Standard deviation of the error between initial focal ratio and the ground truth, unit-less (focal ratio = focal length / image width).
distortionErrorStddev	Standard deviation of the error between initial lens distortion parameters and the ground truth.
rbcErrorStddev	Standard deviation of the error between initial gyro-camera orientation and the ground truth. Measured for rbc in axis-angle representation.
tauGCErrorStddev	Standard deviation of the error between initial gyro-camera time offset and the ground truth, in microseconds.
tauRSSErrorStddev	Standard deviation of the error between initial offset of rolling shutter skew and the ground truth, in microseconds.

2.3 mvCACStatus Struct Reference

```
#include <mvCAC.h>
```

2.3.1 Detailed Description

CAC status.

Parameters:

reprojectionError	Re-projection error in pixels.
inlierRatio	Inlier ratio.

2.4 mvCameraConfiguration Struct Reference

```
#include <mv.h>
```

2.4.1 Detailed Description

Camera calibration parameters. This information could come from any calibration procedure including the CAC feature within this library.

The pixel coordinate space [u, v] has the origin [0, 0] in the upper-left image corner. The u-axis runs towards right along the row in memory address increasing order, and the v-axis runs downward along the column also in memory address increasing order but with a stride length equal to the row width.

The camera coordinate system [x, y, z] is centered on the camera principle point. The positive x-axis of the camera points from the center principle point along that row of pixels [u]. The y-axis points down from the camera center along a column of pixels [v]. The z-axis points directly out along the optical axis in the direction that the camera is pointing.

NOTE: This is the same coordinate system used by OpenCV.

Parameters:

pixelWidth	Width of the image in pixels.
pixelHeight	Height of the image in pixels.
memoryStride	Memory width in bytes to the same pixel one row below.
uvOffset	Optional memory offset to UV plane for NV21 images. Note, this is the U and V color planes of the NV21 format and not to be confused with the u and v axes in image space.
principalPoint[2]	Principal point [u, v] in pixels is defined relative to camera origin in pixel space where [0, 0] is the upper-left image corner, u runs towards right along the row, and v runs downward along the column.
focalLength[2]	Focal length expressed in pixels and as separate components along the image [width, height]. These components are aligned with the [u, v] axes of the principalPoint[2].
distortion	Distortion coefficients. All unused array elements must be set to 0. distortion[0] would be equivalent to k1 in OpenCV or the constant a in the fisheye paper, distortion[1] would be k2 or the constant b in the paper, and so on.
distortionModel	The distortion model is limited to the following values: • **0** = No distortion model • **4** = Four parameter polynomial [k1, k2, p1, p2] plumb-line (a.k.a., Brown-Conrady) model [D. C. Brown, "Photometric Engineering", Vol. 32, No. 3, pp.444-462 (1966)]. Compatible with the oldest Caltech Matlab Calibration Toolbox (`http://www.vision.caltech.edu/bouguetj/calib_doc/`). To fill from OpenCV, declare cv::Mat for distortions with 5 rows (1 columns), set it to zeros and use flag cv::CALIB_FIX_K3 with cv::calibrateCamera. • **5** = Five parameter polynomial [k1, k2, p1, p2, k3] plumb-line model. Compatible with current Matlab toolbox. To fill from OpenCV, declare cv::Mat for distortions with 5 rows, use flag cv::CALIB_FIX_K4 use cv::calibrateCamera. • **8** = Eight parameter rational polynomial (i.e., CV_CALIB_RATIONAL_MODEL) [k1, k2, p1, p2, k3, k4, k5, k6].

	• **10** = FishEye model [S.Shah, "Intrinsic Parameter Calibration Procedure for a (High-Distortion) Fish-eye Lens Camera with Distortion Model and Accuracy Estimation"]. To fill from OpenCV, use cv::fisheye::calibrate.

2.5 mvCameraData Struct Reference

```
#include <mvSRW.h>
```

2.5.1 Detailed Description

Parameters:

desc	Camera descriptor to be used as correspondence with camera name given by frames.
params	Camera parameters.

2.6 mvCameraExtrinsicParameters Struct Reference

```
#include <mvSRW.h>
```

2.6.1 Detailed Description

Parameters:

rbc	Rotation from camera coordinate to body coordinate use by attitude.
timeOffset	Offset between camera and IMU timestamps. IMU timestamp translates to camera timestamp t + timeOffset.
rollingShutterSkew	Rolling shutter skew of the camera, which is the elapsed time from beginning of the first image row to the beginning of the last row.

2.7 mvCPA_Configuration Struct Reference

```
#include <mvCPA.h>
```

2.7.1 Detailed Description

Configuration parameters for initializing mvCPA.

MVCPA_MODE_HISTOGRAM follows the steps below, and stops when desired frame brightness is achieved:

1. Set exposure and gain to minimum
2. Increase gain until hitting soft max
3. Increase exposure until hitting soft max
4. Increase gain until hitting max
5. Increase exposure until hitting max

Parameters:

width	Input image width.
height	Input image height.
format	Input image format.
cpaType	CPA algorithm type.
legacyCost	Parameters for cpaType MVCPA_MODE_LEGACY or MVCPA_MODE_COST.
startExposure	Initial exposure value (normalized to 0.0 - 1.0 range).
startGain	Initial gain value (normalized to 0.0 - 1.0 range).
filterSize	Internal filter size for exposure and gain changes [larger the slower convergence (0 = no filtering)].
gainCost	Cost to increase gain used for cost based approach. Guidelines: gainCost and exposureCost ratio will in the long run be the ratio between gain and exposure values. The sum of gainCost and exposureCost influences how much brightness cost is weight. if gainCost+exposureCost > 1.0, minimizing gain and exposure values is weight higher then then hit brightness goal. If sum < 1.0 brightness goal is more important.
exposureCost	Cost to increase exposure.
enableHistogramCost	Turns on extra saturation protection for cost based algorithm.
thresholdUnderflowed	Allowed brightness margin based on default goal 128 (e.g., with systemBrightnessMargin 30, the brightness goal can be dynamically in [98, 158].
thresholdSaturated	Overexposure threshold on mean brightness of a single block.

systemBrightnessMargin	Underexposure threshold on mean brightness of a single block.
histogram	Parameters for cpaType MVCPA_MODE_HISTOGRAM.
exposureMin	Minimum exposure value (0 < exposureMin). Typically very close to 0, such as 0.001.
exposureSoftMax	Soft maximum exposure value (exposureMin <= exposureSoftMax). Exposure > exposureSoftMax if gain == gainMax. Typically in the low range to minimize motion blur, such as 0.2. This value can potentially be increases for robots limited to slow speeds.
exposureMax	Maximum exposure value (exposureSoftMax <= exposureMax <= 1). Set exposureMax to be either exposureSoftMax or 1. Do the former if you would rather have dark image over blurry image. Do the latter if it's the opposite.
gainMin	Minimum gain value (0 < gainMin). Typically very close to 0, such as 0.001.
gainSoftMax	Soft maximum gain value (gainMin <= gainMax). Gain > gainSoftMax if exposure >= exposureSoftMax. Typically in the low range to reduce noise, such as 0.3. Set gainSoftMax to the maximum gain value which produces acceptable noise (e.g., acceptable denoising artifacts) for your camera.
gainMax	Maximum gain value (gainSoftMax <= gainMax <= 1).
logEGPStepSizeMin	Minimum step size of exposure-gain product adjustment in each update. log2(new_exposure * new_gain) = log2(exposure * gain) + delta 0 < logEGPStepSizeMin <= abs(delta) <= logEGPStepSizeMax Typically very close to 0, such as 0.001. Adjust logEGPStepSizeMax to trade between convergence speed and stability. The default value is 1.0. Larger value converges faster, but may oscillate.
logEGPStepSizeMax	Maximum step size of exposure-gain product adjustment in each update. See logEGPStepSizeMin. Typically around 1.0.

2.8 mvDFSParameters Struct Reference

```
#include <mvDFS.h>
```

2.8.1 Detailed Description

The parameters optionally use to initialize DFS.

Parameters:

textureThreshold	Filters out areas of the image without texture. Range is [0 - 1] with 0 meaning no threshold is applied. A good value is ~0.1.
aggregationWindowWidth	Size of the aggregation window.
aggregationWindowHeight	Size of the aggregation window.
maxSpeckleSize	If a block of pixels with similar disparity is smaller than maxSpeckleSize it will be removed.

2.9 mvDFT_Configuration Struct Reference

```
#include <mvDFT.h>
```

2.9.1 Public Attributes

- int **minNrFeatures**
  camera intrinsic calibration params
- int **maxNrFeatures**

2.9.2 Detailed Description

Configuration parameters for initializing mvDFT.

2.9.3 Member Data Documentation

int maxNrFeatures

minNrFeatures forced as input to optical flow (the fewer, the less stable in texture poor areas)

2.10 mvDFT_Data Struct Reference

```
#include <mvDFT.h>
```

2.10.1 Detailed Description

2D displacement estimate from mvDFT + quality indicators.

2.11 mvFrame Struct Reference

```
#include <mvSRW.h>
```

2.11.1 Detailed Description

Camera frame.

Parameters:

timestamp	Timestamp of data in microseconds. Time must be center of exposure time and not the start or end of frame.
leftImage	This is the only image in the monocular case. In the stereo case, this is the left image.
rightImage	In the stereo case, this is the right image. In the monocular case, it is invalid.

2.12 mvGPStimeSyncData Struct Reference

```
#include <mvSRW.h>
```

2.12.1 Detailed Description

GPS time sync data.

Parameters:

timestamp	Timestamp of data in microseconds.
bias	Value for the time bias/offset between GPS and IMU (system) clocks.
GPStimeUncertaintyStd	GPS time uncertainty.

2.13 mvGPSvelocityData Struct Reference

```
#include <mvSRW.h>
```

2.13.1 Detailed Description

GPS velocity data.

Parameters:

timestamp	GPS Timestamp of data in picoseconds.
x	Velocity for the x-axis.
y	Velocity for the y-axis.
z	Velocity for the z-axis.

2.14 mvImage Struct Reference

```
#include <mvSRW.h>
```

2.14.1 Detailed Description

Image data structure.

Parameters:

pixels	Pointer to 8-bit grayscale image luminance data.
width	Width of image in pixels.
height	Height of image in pixels.
memoryStride	Number of bytes to pixel directly one row below.

2.15 mvIMUData Struct Reference

```
#include <mvSRW.h>
```

2.15.1 Detailed Description

IMU data structure.

Parameters:

timestamp	Timestamp of data in microseconds.
x	Value for the x-axis.
y	Value for the y-axis.
z	Value for the z-axis.

2.16 mvPose3DR Struct Reference

```
#include <mv.h>
```

2.16.1 Detailed Description

3-DOF pose information in rotation matrix form.

Parameters:

matrix	Rotation matrix [R] in row major order.

2.17 mvPose6DET Struct Reference

```
#include <mv.h>
```

2.17.1 Detailed Description

Pose information in Euler-Translation form.

Parameters:

translation[3]	Translation vector in use defined units.
euler[3]	Euler angles in the Tait-Bryan ZYX convention. euler[0] = rotation about x-axis. euler[1] = rotation about y-axis. euler[2] = rotation about z-axis (defined from y-axis).

2.18 mvPose6DRT Struct Reference

```
#include <mv.h>
```

2.18.1 Detailed Description

6-DOF pose information in Rotation-Translation matrix form.

Parameters:

matrix	[R | T] rotation matrix + translation column vector in row major order.

2.19 mvSACConfiguration Struct Reference

```
#include <mvSAC.h>
```

2.19.1 Detailed Description

Configuration parameters for initializing mvSAC.

Parameters:

maxNumKeypoints	Maximum number of key points used for tracking.

2.20 mvSACStatus Struct Reference

```
#include <mvSAC.h>
```

2.20.1 Detailed Description

SAC status.

Parameters:

reprojectionError	Re-projection error in pixels.
inlierRatio	Inlier ratio.

2.21 mvStereoConfiguration Struct Reference

```
#include <mv.h>
```

2.21.1 Detailed Description

Stereo rig configuration. This information could come from any calibration procedure including the SAC feature within this library. The cameras in the Qualcomm Flight stereo kit are laid out in such a way as when looking from behind the cameras and into the direction that the camera points, the left camera is camera[0] and the right camera is camera[1]. The camera coordinate systems are described in the **mvCameraConfiguration** description.

The rig coordinate system is aligned with the camera[0] coordinate system. The positive x-axis is aligned with the camera[0] u-axis but would also be fairly close to the line between the centers of camera[0] and camera[1] for the Qualcomm Flight stereo kit. This is the same coordinate system used by OpenCV.

See the `Snapdragon::DfsRosNode::InitDfs()` example in `https://github.qualcomm.com/ATLFlight/dfs-ros-example/blob/develop/src/nodes/SnapdragonDfsRos.cpp` for and example of going from ROS calibration parameters to MV parameters.

Parameters:

translation[3]	Relative distance in meters added to a point from camera[1] in rig coordinates to align to the same point in camera[0]. Therefore translation[0] is usually a negative number nearly equal to the baseline value for the Qualcomm Flight stereo kit since camera[1] is approximately the baseline value away along the rig coordinates x-axis. Same as self.T from ROS camera calibration tool and same as T from OpenCV cvStereoCalibrate() function. • translation[0] = x-axis translation. • translation[1] = y-axis translation. • translation[2] = z-axis translation (defined from the x-y plane).
rotation[3]	Relative rotation between cameras. The rotation is a scaled axis-angle vector representation of the rotation between the two cameras also known as the Rodrigues' rotation formula in the aforementioned rig coordinate system. See `https://jsfiddle.net/1gej4qyp/` for example of converting a rotation matrix to scales-axis representation. Same as

	R from OpenCV cvStereoCalibrate() function. The ROS calibration tool output self.R would be the input rotation matrix to the Rodrigues' formula.
camera[2]	Left/right camera calibrations.
correctionFactors[4]	Polynomial coefficients for a distance-to-distance correction function.

2.22 mvTrackingPose Struct Reference

```
#include <mv.h>
```

2.22.1 Detailed Description

Pose information along with a quality indicator.

Parameters:

pose	6-DOF pose.
poseQuality	Quality of the pose.

2.23 mvVISLAMMapPoint Struct Reference

```
#include <mvVISLAM.h>
```

2.23.1 Public Types

- enum **QUALITY_T** { **LOW**, **MEDIUM**, **HIGH** }

2.23.2 Detailed Description

Map point information from VISLAM.

Parameters:

id	Unique ID for map point.
pixLoc	2D measured pixel location in pixels.
tsf	3D location in spatial frame in meters.

p_tsf	Error covariance for tsf.
depth	Depth of map point from camera in meters.
depthErrorStdDev	Depth error standard deviation in meters.
pointQuality	Quality of the map point as per current VISLAM state.

2.23.3 Member Enumeration Documentation

enum QUALITY_T

Enumerator:

LOW	additional low-quality points collected for e.g. collision avoidance
MEDIUM	Points that are not "in state".
HIGH	Points that are "in state".

2.24 mvVISLAMPose Struct Reference

```
#include <mvVISLAM.h>
```

2.24.1 Detailed Description

Pose information along with a quality indicator for VISLAM.

Parameters:

poseQuality	Quality of the pose (no pose is provided if MV_TRACKING_STATE_INITIALIZING or MV_TRACKING_STATE_FAILED). If the IMU measurement range or bandwidth is exceeded, MV_TRACKING_STATE_LOW_QUALITY is returned. In normal operation, poseQuality should correspond to MV_TRACKING_STATE_HIGH_QUALITY.
bodyPose	Body pose estimate in rotation-translation matrix form [R_{sb} | T_{sb}]. T_{sb} is the estimate of the translation of the origin of the body (b) or accelerometer frame relative to the spatial (s) frame

	in the spatial frame (meters). The spatial frame corresponds to the body frame at initialization. R_{sb} is the estimate of the corresponding rotation matrix.
gravityCameraPose	Gravity aligned pose of camera estimate in rotation-translation matrix form [R_{cs'} | T_{cs'}]. T_{cs'} is the estimate of the translation of the origin of the camera frame (c) relative to the gravity aligned spatial (s') frame in the camera frame (meters). The gravity aligned spatial frame corresponds to the body frame at initialization rotated to compensate for pitch and roll. R_{cs'} is the estimate of the corresponding rotation matrix.
errCovPose	Error covariance matrix for bodyPose estimate.
timeAlignment	Camera IMU time misalignment estimate (seconds).
velocity	Velocity estimate, vsb, of origin of accelerometer (b=body) in spatial frame (m/s). The spatial frame corresponds to the body frame at initialization.
errCovVelocity	Error covariance for velocity estimate vsb ((m/s)^2).
angularVelocity	Angular velocity estimate in body (accelerometer) frame (rad/s).
gravity	Gravity vector estimate in spatial frame (m/s^2). The spatial frame corresponds to the body frame at initialization.
errCovGravity	Error covariance for gravity estimate in spatial frame ((m/s^2)^2).
wBias	Gyro bias estimate (rad/s).
aBias	Accelerometer bias estimate (m/s^2).
Rbg	Accelerometer gyro rotation matrix estimate (b = body = accelerometer, g = gyro).
aAccInv	Inverse of accelerometer scale and non-orthogonality estimate.
aGyrInv	Inverse of gyro scale and non-orthogonality estimate.
tbc	Accelerometer-camera translation misalignment vector estimate (meters). t_{bc} is the estimate of the translation of the origin of the camera (c) frame relative to that of the body (b) or accelerometer frame in the body frame.
Rbc	Accelerometer-camera rotational misalignment matrix estimate. Can be used together with tbc to rotate vector in camera frame x_c to IMU frame x_imu via [Rbc|tbc]x_c = x_imu.
errorCode	Error code (includes reasons for reset) bit: • **0** : Reset: cov not pos definite • **1** : Reset: IMU exceeded range • **2** : Reset: IMU bandwidth too low • **3** : Reset: not stationary at initialization • **4** : Reset: no features for x seconds • **5** : Reset: insufficient constraints from features • **6** : Reset: failed to add new features

	• **7** : Reset: exceeded instant velocity uncertainty • **8** : Reset: exceeded velocity uncertainty over window • **10** : Dropped IMU samples • **11** : Intrinsic camera cal questionable • **12** : Insufficient number of good features to initialize • **13** : Dropped camera frame • **14** : Dropped GPS velocity sample • **15** : Sensor measurements with uninitialized time stamps or uninitialized uncertainty (set to 0) If a reset occurs, the last "good" pose will be used after initialization. To reset the pose, call **mvVISLAM_Reset(mvVISLAM* pObj, bool resetPose)** with resetPose set to true.
time	Timestamp of pose in nanoseconds in system time.

2.25 mvVM_CollisionInfo Struct Reference

```
#include <mvVM.h>
```

2.25.1 Detailed Description

Return data for collision checking and distance computation functions.

Parameters:

point	3D sample point location in meters that collided, this is only valid for collision types MV_COLLISION_YES and MV_COLLISION_UNKNOWN.
type	Return value of the collision detection.

2.26 mvVM_IntegrationConfiguration Struct Reference

```
#include <mvVM.h>
```

2.26.1 Public Types

- enum { **SURFACE**, **VISIBLE**, **EXISTING_VISIBLE** }

2.26.2 Detailed Description

Configuration structure for integration functions.

Parameters:

noise	Noise associated with range measurements.
filterWeight	Filter delay, essentially the length of the moving average filter.
updateMode	What parts of the voxel grid are updated with new range data.

2.26.3 Member Enumeration Documentation

anonymous enum

Enumerator:

SURFACE	only update around measurements
VISIBLE	update whole visible frustum
EXISTING_VISIBLE	update existing voxels in the visible frustum

3 File Documentation

3.1 mv.h File Reference

```
#include <stddef.h>
#include <stdbool.h>
#include <stdint.h>
```

3.1.1 Classes

- struct **mvCameraConfiguration**
- struct **mvStereoConfiguration**
- struct **mvPose3DR**
- struct **mvPose6DRT**
- struct **mvPose6DET**
- struct **mvTrackingPose**

3.1.2 Enumerations

- enum **MV_TRACKING_STATE**
- enum **MV_COLLISION** : int32_t { **MV_COLLISION_NO** = 0, **MV_COLLISION_YES** = 1, **MV_COLLISION_UNKNOWN** = 2 }

3.1.3 Functions

- const char * **mvVersion** (void)
- void **mvPose6DETto6DRT** (**mvPose6DET** *pose, **mvPose6DRT** *mvPose)
- void **mvPose6DRTto6DET** (**mvPose6DRT** *pose, **mvPose6DET** *mvPose)
- void **mvMultiplyPose6DRT** (const **mvPose6DRT** *A, const **mvPose6DRT** *B, **mvPose6DRT** *out)
- void **mvInvertPose6DRT** (**mvPose6DRT** *pose)
- void **mvGetGLProjectionMatrix** (**mvCameraConfiguration** *camera, float64_t nearClip, float64_t farClip, float64_t *mat, bool transpose)
- void **mvPoseAngles** (**mvPose6DRT** *pose, float *yaw, float *pitch, float *roll)

3.1.4 Detailed Description

mv.h

Common data structures and utilities for the Machine Vision SDK.

3.1.5 Enumeration Type Documentation

enum MV_COLLISION : int32_t

Return values for collision detection functions.

Enumerator:

MV_COLLISION_NO	no collision occurred
MV_COLLISION_YES	a collision was found
MV_COLLISION_UNKNOWN	unmapped area was found

enum MV_TRACKING_STATE

Tracking state quality.

3.1.6 Function Documentation

void mvGetGLProjectionMatrix (mvCameraConfiguration * *camera*, float64_t *nearClip*, float64_t *farClip*, float64_t * *mat*, bool *transpose*)

OpenGL helper function.

Parameters:

transpose	Flag of whether transpose is needed.

void mvInvertPose6DRT (mvPose6DRT * *pose*)

Invert mvPose6RT in place, computes pose = pose^-1

void mvMultiplyPose6DRT (const mvPose6DRT * *A*, const mvPose6DRT * *B*, mvPose6DRT * *out*)

Multiply two **mvPose6DRT**, computes out = A * B

void mvPose6DETto6DRT (mvPose6DET * *pose*, mvPose6DRT * *mvPose*)

Convert Euler-Translation pose to Rotation-Translation.

void mvPose6DRTto6DET (mvPose6DRT * *pose*, mvPose6DET * *mvPose*)

Convert Rotation-Translation pose to Euler-Translation. Follows Tait-Bryan convention so that:

euler[0] = rotation about x-axis.

euler[1] = rotation about y-axis.

euler[2] = rotation about z-axis (defined from y-axis).

void mvPoseAngles (mvPose6DRT * *pose*, float * *yaw*, float * *pitch*, float * *roll*)

Get Yaw, Pitch, and Roll of camera pose in target coordinate system (Z up, Y right, X out of target and camera system is x right, y down and z out of camera).

Parameters:

pose	Pose to calculate angles from.
yaw	Results of yaw calculation, rotation of x axis direction y (in x/y plane) (target coordinates).
pitch	Results of pitch calculation, rotation of z axis direction x (in z/x plane) (target coordinates).
roll	Results of roll calculation, rotation of z axis direction y (in z/y plane) (target coordinates).

const char* mvVersion (void)

Return string of version information.

3.2 mvCAC.h File Reference

```
#include <mv.h>
```

3.2.1 Classes

- struct **mvCACConfiguration**
- struct **mvCACStatus**

3.2.2 Typedefs

- typedef struct **mvCAC** **mvCAC**

3.2.3 Functions

- **mvCAC** * **mvCAC_Initialize** (const **mvCameraConfiguration** *pCamCfg, const uint8_t *mask, uint32_t maskStride, const **mvPose3DR** *pRbc, const **mvCACConfiguration** *pCACCfg)
- void **mvCAC_Deinitialize** (**mvCAC** *pObj)
- void **mvCAC_AddGyro** (**mvCAC** *pObj, int64_t timestamp, const float64_t x, const float64_t y, const float64_t z)
- void **mvCAC_AddFrame** (**mvCAC** *pObj, int64_t timestamp, int64_t rollingShutterSkew, const uint8_t *pixels, uint32_t stride)
- void **mvCAC_AddTrackedPoints** (**mvCAC** *pObj, int64_t timestamp, int64_t rollingShutterSkew, const float32_t *pts1, const float32_t *pts2, uint32_t numPts)
- **MV_TRACKING_STATE** **mvCAC_GetCalibration** (**mvCAC** *pObj, **mvCameraConfiguration** *pCfg, **mvPose3DR** *pRbc, float64_t *tauGyroCamera, float64_t *tauRollingShutterSkew, **mvCACStatus** *pStatus)
- float64_t **mvCAC_FisheyeToPolynomial** (**mvCameraConfiguration** *pCfg, int32_t model)
- float32_t **mvCAC_ScoreSceneTexture** (const uint8_t *pixels, uint32_t width, uint32_t height, uint32_t stride)

3.2.4 Detailed Description

mvCAC.h

Machine Vision, Camera Auto-Calibration (CAC)

3.3 Overview

This module performs mono camera auto-calibration, which does not require a known pattern in front of the camera. It calibrates the following parameters: camera intrinsic, lens distortion, camera-gyro orientation, camera-gyro time offset.

3.4 Limitations

CAC may produce incorrect results if the following conditions are not met:

- Exposure time must be shorter than 5ms.
- Adjacent frames must have similar brightness level. It is recommended to use constant exposure and gain, if AEC cannot meet this requirement.

3.5 Recommendations

CAC converges in 900 frames if the following conditions are met:

- Camera rotates around all 3 axis.
- Median inter-frame camera rotation is at least 2 degrees.
- 99% of inter-frame camera rotation is no more than 6 degrees.
- At least 80% of the scene is textured, and at least 3 meters away.
- Inter-frame camera translation is no more than 1cm.

- Restriction on camera translation can be relaxed in proportion to scene distance. For example, if 80% of the scene is 30 meters away, CAC can tolerate inter-frame camera translation up to 10cm.

3.5.1 Typedef Documentation

typedef struct mvCAC mvCAC

Camera Auto-Calibration (CAC)

3.5.2 Function Documentation

void mvCAC_AddFrame (mvCAC * *pObj*, int64_t *timestamp*, int64_t *rollingShutterSkew*, const uint8_t * *pixels*, uint32_t *stride*)

Add camera frame.

This function performs feature tracking internally. Call mvCAC_AddTrackedPoints instead for external tracking.

Parameters:

pObj	Pointer to CAC object.
timestamp	Timestamp of the first row at the center of exposure, in microseconds.
rollingShutterSkew	The duration between the start of first row exposure and the start of last row exposure, in microseconds.
pixels	Pointer to the pixels of luma channel.
stride	Stride of the luma channel in bytes.

void mvCAC_AddGyro (mvCAC * *pObj*, int64_t *timestamp*, const float64_t *x*, const float64_t *y*, const float64_t *z*)

Add gyro measurement

Gyro measurements must be added in chronological order. All measurements received before the end of the frame must be added before either mvCAC_AddFrame or mvCAC_AddTrackedPoints is called.

Parameters:

pObj	Pointer to CAC object.
timestamp	Timestamp of the gyro measurement, in microseconds.
x	Gyro measurement for X axis, in rad/s.
y	Gyro measurement for Y axis, in rad/s.
z	Gyro measurement for Z axis, in rad/s.

void mvCAC_AddTrackedPoints (mvCAC * *pObj*, int64_t *timestamp*, int64_t *rollingShutterSkew*, const float32_t * *pts1*, const float32_t * *pts2*, uint32_t *numPts*)

Add tracked points in a camera frame.

This function allows the caller to do its own feature tracking. If the caller doesn't have one, call mvCAC_AddFrame instead. The feature tracker should have tracking error less than half of a pixel, and less than 10% outliers.

Parameters:

pObj	Pointer to CAC object.
timestamp	Timestamp of the first row at the center of exposure, in microseconds.
rollingShutterSkew	The duration between the start of first row exposure and the start of last row exposure, in microseconds.
pts1	Tracked 2D points in the previous frame. X, Y coordinates are stored as $ (x_k, y_k) = (pts1[k*2], pts1[k*2+1]). k = 0 .. numPts-1 $
pts2	Tracked 2D points in the current frame.
numPts	Number of tracked points.

void mvCAC_Deinitialize (mvCAC * *pObj*)

Deinitialize Camera Auto-Calibration (CAC) object.

Parameters:

pmObj	Pointer to CAC object.

float64_t mvCAC_FisheyeToPolynomial (mvCameraConfiguration * *pCfg*, int32_t *model*)

Convert fisheye model to polynomial model.

See **mv.h** for more details on distortion model.

Parameters:

pCfg	Pointer to the camera parameters to be converted and updated. The input distortion model must be 10, otherwise no conversion will be performed.
model	Desired distortion model, which can be either 4, 5, or 8. No conversion will be performed if the specified model is not allowed.

Returns:

RMSE between output and input model, in pixels. Returns 0 if conversion is not performed.

MV_TRACKING_STATE mvCAC_GetCalibration (mvCAC * *pObj*, mvCameraConfiguration * *pCfg*, mvPose3DR * *pRbc*, float64_t * *tauGyroCamera*, float64_t * *tauRollingShutterSkew*, mvCACStatus * *pStatus*)

Get the calibration result.

Parameters:

pObj	Pointer to CAC object.
pCfg	Calibrated intrinsic parameters. Set to nullptr if not needed. The distortion model is always fisheye (model 10).
pRbc	Calibrated Rbc. Set to nullptr if not needed.
tauGyroCamera	Calibrated time offset from camera to gyroscope, in microseconds. Set to nullptr if not needed.
tauRollingShutterSkew	Calibrated offset of rolling shutter skew, in microseconds. Set to nullptr if not needed.
td	Calibrated time offset between camera and attitude, in microseconds. Set to nullptr if not needed.
pStatus	CAC status. Set to nullptr if not needed.

Returns:

Tracking state. No calibration result is returned if state < 0.

mvCAC* mvCAC_Initialize (const mvCameraConfiguration * *pCamCfg*, const uint8_t * *mask*, uint32_t *maskStride*, const mvPose3DR * *pRbc*, const mvCACConfiguration * *pCACCfg*)

Initialize Camera Auto-Calibration (CAC) object.

Parameters:

pCamCfg	Initial values for camera calibration parameters. memoryStride and uvOffset are ignored.
mask	Mask of good camera pixels. Its size is the same as camera frame size. 0 means the corresponding camera pixel should be ignored by CAC. >0 means the corresponding camera pixel should be processed by CAC. If mask == nullptr, CAC will process all camera pixels, as if all mask pixels > 0.
maskStride	Stride of mask in bytes. Ignored when mask == nullptr.
pRbc	Initial value for rotation from camera coordinate frame to body coordinate frame. It is assumed that the gyroscope frame is the same as the body frame.
pCACCfg	CAC configuration parameters.

Returns:

Pointer to CAC object; returns NULL if failed.

float32_t mvCAC_ScoreSceneTexture (const uint8_t * *pixels*, uint32_t *width*, uint32_t *height*, uint32_t *stride*)

Score the scene for textureness.

Parameters:

pixels	Pointer to the pixels of luma channel.
width	Image width.
height	Image height.
stride	Image stride in bytes.

Returns:

Score between 0 and 1. Higher score means more texture in the scene.

3.6 mvCPA.h File Reference

```
#include <mv.h>
```

3.6.1 Classes

- struct **mvCPA_Configuration**

3.6.2 Typedefs

- typedef struct **mvCPA mvCPA**

3.6.3 Enumerations

- enum **MVCPA_MODE**
- enum **MVCPA_FORMAT**

3.6.4 Functions

- **mvCPA** * **mvCPA_Initialize** (const **mvCPA_Configuration** *cpaConfig)
- void **mvCPA_Deinitialize** (**mvCPA** *pObj)
- void **mvCPA_AddFrame** (**mvCPA** *pObj, const uint8_t *pixels, uint32_t stride)
- void **mvCPA_GetValues** (**mvCPA** *pObj, float32_t *exposure, float32_t *gain)

3.6.5 Detailed Description

mvCPA.h

Machine Vision, Camera Parameter Adjustment (CPA)

3.7 Overview

CPA provides changes to camera parameters for online auto gain and exposure control.

3.8 Limitations

The following list are some of the known limitations:

- Only designed and tested with OV7251 based camera modules.

3.8.1 Typedef Documentation

typedef struct mvCPA mvCPA

Camera Parameter Adjustment (CPA)

3.8.2 Enumeration Type Documentation

enum MVCPA_FORMAT

CPA image format.

- MVCPA_FORMAT_GRAY8: 8-bit grayscale format.
- MVCPA_FORMAT_RAW10: Android 10-bit raw format.
- MVCPA_FORMAT_RAW12: Android 12-bit raw format.

enum MVCPA_MODE

CPA algorithm mode.

- MVCPA_MODE_LEGACY: Unlikely to be the best choice for any use case.
- **WARNING:** to be deprecated.
- MVCPA_MODE_COST: A good trade off of illumination for viewable images while still favoring computer vision needs over illumination.
- MVCPA_MODE_HISTOGRAM: Most focused towards computer vision needs and best at supporting higher speeds of camera movement.

3.8.3 Function Documentation

void mvCPA_AddFrame (mvCPA * *pObj*, const uint8_t * *pixels*, uint32_t *stride*)

Add image to adjust exposure and gain parameters on. (Assumption is that this was taking with last returned parameters).

Parameters:

pObj	Pointer to CPA object.
pixels	Pointer to Luminance pixels of camera frame.
width	Width of the given frame data.
height	Height of the given frame data.
stride	Stride of the given frame data.

void mvCPA_Deinitialize (mvCPA * *pObj*)

Deinitialize Camera Parameter Adjustment (CPA) object.

Parameters:

pmObj	Pointer to CPA object.

void mvCPA_GetValues (mvCPA * *pObj*, float32_t * *exposure*, float32_t * *gain*)

Access estimated exposure and gain values.

Parameters:

pObj	Pointer to CPA object.
exposure	Pointer to returned new exposure value estimation.
gain	Pointer to returned new gain values estimation.

mvCPA* mvCPA_Initialize (const mvCPA_Configuration * *cpaConfig*)

Initialize Camera Parameter Adjustment (CPA) object.

Parameters:

cpaConfig	Configuration parameters to initialize CPA.

Returns:

Pointer to CPA object; returns NULL if failed.

3.9 mvDFS.h File Reference

`#include <mv.h>`

3.9.1 Classes

- struct **mvDFSParameters**

3.9.2 Typedefs

- typedef struct **mvDFS mvDFS**

3.9.3 Enumerations

- enum **MVDFS_MODE**

3.9.4 Functions

- **mvDFS** * **mvDFS_Initialize** (const **mvStereoConfiguration** *nConfig, **MVDFS_MODE** mode, bool using10bitInput, const **mvDFSParameters** *params=NULL)
- void **mvDFS_Deinitialize** (**mvDFS** *pObj)
- void **mvDFS_GetDepths** (**mvDFS** *pObj, const uint8_t *pxlsCamL, const uint8_t *pxlsCamR, uint16_t numMasks, uint16_t *masks, int16_t minDisparity, int16_t maxDisparity, uint16_t *disparities, float32_t *invDepth)
- void **mvDFS_GetDepthsION** (**mvDFS** *pObj, int fileDesc, void *hostPtr, size_t bufSize, uint16_t numMasks, uint16_t *masks, int16_t minDisparity, int16_t maxDisparity, uint16_t *disparities, float32_t *invDepth)
- void **mvDFS_GetRectifyingRotations** (**mvDFS** *obj, float32_t *rot1, float32_t *rot2)
- void **mvDFS_GetDepthCameraConfiguration** (**mvDFS** *obj, **mvCameraConfiguration** *depthCamera)
- void **mvDFS_GetRectifiedImages** (**mvDFS** *obj, uint8_t *rectL, uint8_t *rectR)

- void **mvDFS_EnableRectAdjustment** (**mvDFS** *obj, float *params, unsigned int numParams)
- void **mvDFS_DisableRectAdjustment** (**mvDFS** *obj)

3.9.5 Detailed Description

mvDFS.h

Machine Vision, Depth from Stereo (DFS)

3.10 Overview

DFS finds the disparity pixels as the x-axis distance (in pixels) of one place in the left image verses that same place in the right image. The assumption of a stereo configuration of the cameras is leveraged for speed. Therefore, this feature is not good for general feature matching. The disparities are mapped directly to the distance away from the camera. A disparity value of 0 would mean the object is at infinity whereas a disparity value of 28 would mean that the object is very close.

There are two algorithms supported for flexibility on whether to use the CPU or GPU. However, the ALG1 on GPU is the primary and preferred algorithm. Although FPS speeds much greater are possible, a typical configuration supporting 30 FPS for MVDFS_MODE_ALG1_GPU is:

```
resolution = QVGA
minDisparity = 0
maxDisparity = 28  // detectable distance = 0.6m for focal length ~217 pel
aggregationWindowSize = 11
```

3.11 Limitations

The following list are some of the known limitations:

- Cannot resolve depths > ~100*(distance between cameras).
- Cannot resolve depth where the field of view does not overlap between both cameras.
- Does not detect transparent, reflective, shiny smooth solid color, overly illuminated, or inadequately illuminated surfaces.
- Does not detect some surfaces with repeating patterns.
- Rig calibration must be good to < 0.5 pixels projection error between left and right images.
- Does not detect linear object (e.g., power line) that run along same rows in images.

3.11.1 Typedef Documentation

typedef struct mvDFS mvDFS

Depth from Stereo (DFS).

3.11.2 Enumeration Type Documentation

enum MVDFS_MODE

Two different algorithms are currently supported on CPU and one on GPU.

- MVDFS_MODE_ALG0_CPU: Lower quality algorithm but very fast on CPU.

- MVDFS_MODE_ALG1_CPU: Higher quality algorithm but very slow on CPU. This mode is primarily for off-target testing since it is too slow for practical use.
- MVDFS_MODE_ALG1_GPU: Higher quality algorithm and very fast on GPU.

3.11.3 Function Documentation

void mvDFS_Deinitialize (mvDFS * *pObj*)

Deinitialize stereo object.

Parameters:

pObj	Pointer to stereo object.

void mvDFS_DisableRectAdjustment (mvDFS * *obj*)

Disables rectification adjustment.

Parameters:

obj	Pointer to stereo object.

void mvDFS_EnableRectAdjustment (mvDFS * *obj*, float * *params*, unsigned int *numParams*)

Enables rectification adjustment and provides the required parameters.

Parameters:

obj	Pointer to stereo object.
params	Pointer to buffer containing the rectification adjustment parameters.
numParams	Number of rectification adjustment parameters.

void mvDFS_GetDepthCameraConfiguration (mvDFS * *obj*, mvCameraConfiguration * *depthCamera*)

Depth camera. This virtual depth camera is obtained during solving the rectification problem.

Parameters:

obj	Pointer to stereo object.
depthCamera	Pointer to camera structure.

void mvDFS_GetDepths (mvDFS * *pObj*, const uint8_t * *pxlsCamL*, const uint8_t * *pxlsCamR*, uint16_t *numMasks*, uint16_t * *masks*, int16_t *minDisparity*, int16_t *maxDisparity*, uint16_t * *disparities*, float32_t * *invDepth*)

Compute inverse depth.

Parameters:

pObj	Pointer to stereo object.

pxlsCamL	Left camera image.
pxlsCamR	Right camera image.
numMasks	Number of rectangular masks.
masks	Mask defined as rectangular region in the depth image in which disparities and depth are to be masked out (set to 0). A single region is defined by four integers being image coordinates of upper left and bottom right corners of the region; if NULL no masking is done.
minDisparity	Lower limit of the disparity range to be scanned. **NOTE:** Should be multiple of 4 for optimal speeds.
maxDisparity	Upper limit of the disparity range to be scanned. **NOTE:** Should be multiple of 4 for optimal speeds.
disparities	Optional disparity for each pixel in the left camera image. Caller allocates and provides buffer with dimensions of camera image.
invDepth	Optional depth for each pixel of left camera image in units 1/meters. Caller allocates the buffer of the size of camera image. Returned 0 values mean depth for given pixel is unknown.

Remarks:

Inverse depth is computed for pixels of rectified left image which is rotated w.r.t. the original left image by rectifying rotation To get rectifying rotations use function mvDFS_getRectifyingRotations.

void mvDFS_GetDepthsION (mvDFS * *pObj*, int *fileDesc*, void * *hostPtr*, size_t *bufSize*, uint16_t *numMasks*, uint16_t * *masks*, int16_t *minDisparity*, int16_t *maxDisparity*, uint16_t * *disparities*, float32_t * *invDepth*)

Compute inverse depth from left and right images stored side-by-side in an ION memory buffer.

Parameters:

pObj	Pointer to stereo object.
fileDesc	ION memory file descriptor.
hostPtr	Host virtual address to ION memory buffer. The GPU requires this to be aligned to the device page size.
bufSize	Size in bytes of the allocation ION buffer.
numMasks	Number of rectangular masks.
masks	Mask defined as rectangular region in the depth image in which disparities and depth are to be masked out (set to 0). A single region is defined by four integers being image coordinates of upper left and bottom right corners of the region; if NULL no masking is done.
minDisparity	Lower limit of the disparity range to be scanned.

	NOTE: Should be multiple of 4 for optimal speeds.
maxDisparity	Upper limit of the disparity range to be scanned. **NOTE:** Should be multiple of 4 for optimal speeds.
disparities	Optional disparity for each pixel in the left camera image. Caller allocates and provides buffer with dimensions of camera image.
invDepth	Optional depth for each pixel of left camera image in units 1/meters. Caller allocates the buffer of the size of camera image. Returned 0 values mean depth for given pixel is unknown.

Remarks:

Inverse depth is computed for pixels of rectified left image which is rotated w.r.t. the original left image by rectifying rotation. To get rectifying rotations use function mvDFS_getRectifyingRotations.

void mvDFS_GetRectifiedImages (mvDFS * *obj*, uint8_t * *rectL*, uint8_t * *rectR*)

Returns rectified left and right gray scale image.

Parameters:

obj	Pointer to stereo object.
rectL	Pointer to rectified left image.
rectR	Pointer to rectified right image.

void mvDFS_GetRectifyingRotations (mvDFS * *obj*, float32_t * *rot1*, float32_t * *rot2*)

Rectification rotation matrices for left and right images as 3x3 rotation matrices.

Parameters:

obj	Pointer to stereo object.
rot1	Pointer to 3x3 matrix in which the rotation of left image returned.
rot1	Pointer to 3x3 matrix in which the rotation of left image returned.

mvDFS* mvDFS_Initialize (const mvStereoConfiguration * *nConfig*, MVDFS_MODE *mode*, bool *using10bitInput*, const mvDFSParameters * *params* = NULL)

Initialize stereo object.

Parameters:

pnConfig	Pointer to configuration.
mode	Select which mode DFS algorithm should run in, i.e quality vs speed vs GPU vs GPU simulation.
using10BitInput	

	true: Input images are 10 bits grayscale with 2 bytes / pixel. false: Input images are 8 bits grayscale with 1 byte / pixel.

Returns:

Pointer to stereo object; returns NULL if failed.

3.12 mvDFT.h File Reference

`#include <mv.h>`

3.12.1 Classes

- struct **mvDFT_Configuration**
- struct **mvDFT_Data**

3.12.2 Typedefs

- typedef struct **mvDFT mvDFT**

3.12.3 Functions

- **mvDFT** * **mvDFT_Initialize** (const **mvDFT_Configuration** *nConfig)
- void **mvDFT_Deinitialize** (**mvDFT** *pObj)
- void **mvDFT_AddImage** (**mvDFT** *pObj, int64_t time, const uint8_t *pxls)
- bool **mvDFT_GetResult** (**mvDFT** *pObj, **mvDFT_Data** *data)

3.12.4 Detailed Description

mvDFT.h

Machine Vision, Downward Facing Tracker (mvDFT)

3.13 Overview

This feature provides frame-by-frame localization for cameras facing mostly straight down.

3.14 Limitations

The following list are some of the known limitations:

- Does not work over transparent, reflective, shiny smooth solid color, overly illuminated, or inadequately illuminated surfaces.
- Does not work over some surfaces with repeating patterns.
- Camera calibration must be good to < 0.5 pixels re-projection error.
- Velocity must be < 50 pixels/frame.

3.14.1 Typedef Documentation

typedef struct mvDFT mvDFT

Downward Facing Tracker (mvDFT).

3.14.2 Function Documentation

void mvDFT_AddImage (mvDFT * *pObj*, int64_t *time*, const uint8_t * *pxls*)

Pass camera frame to the mvDFT object.

Parameters:

pObj	Pointer to mvDFT object.
time	Timestamp of camera frame.
pxls	Pointer to camera frame data.

void mvDFT_Deinitialize (mvDFT * *pObj*)

Deinitialize mvDFT object.

Parameters:

pObj	Pointer to mvDFT object.

bool mvDFT_GetResult (mvDFT * *pObj*, mvDFT_Data * *data*)

Displacement data.

Parameters:

pObj	Pointer to mvDFT object.
data	Pointer to **mvDFT_Data** data array.

Returns:

Success or not.

mvDFT* mvDFT_Initialize (const mvDFT_Configuration * *nConfig*)

Initialize mvDFT object.

Parameters:

pnConfig	Pointer to configuration.

Returns:

Pointer to mvDFT object; returns NULL if failed.

3.15 mvSAC.h File Reference

`#include <mv.h>`

3.15.1 Classes

- struct **mvSACConfiguration**
- struct **mvSACStatus**

3.15.2 Typedefs

- typedef struct **mvSAC** **mvSAC**

3.15.3 Functions

- **mvSAC** * **mvSAC_Initialize** (const **mvCameraConfiguration** *pCfgL, const **mvCameraConfiguration** *pCfgR, const float32_t translation[3], const **mvSACConfiguration** *pSACCfg)
- void **mvSAC_Deinitialize** (**mvSAC** *pObj)
- void **mvSAC_AddFrame** (**mvSAC** *pObj, const uint8_t *pixelsL, uint32_t strideL, const uint8_t *pixelsR, uint32_t strideR)
- **MV_TRACKING_STATE** **mvSAC_GetCalibration** (**mvSAC** *pObj, **mvStereoConfiguration** *pStereoCfg, **mvSACStatus** *pStatus)

3.15.4 Detailed Description

mvSAC.h

Machine Vision public API, Stereo Auto-Calibration (SAC)

3.16 Overview

This module performs stereo camera auto-calibration, which does not require a known pattern in front of the camera. It calibrates the following parameters: rotation from left camera to right.

3.17 Limitations

The following list are some of the known limitations:

- Requires textured objects in front of the camera for tracking; Otherwise SAC will not return any result.
- Exposure time must be shorter than 5ms; Otherwise SAC may return incorrect results.
- Typically needs at least 3 seconds of data to produce good quality results.

3.17.1 Typedef Documentation

typedef struct mvSAC mvSAC

Stereo Auto-Calibration (SAC).

3.17.2 Function Documentation

void mvSAC_AddFrame (mvSAC * *pObj*, const uint8_t * *pixelsL*, uint32_t *strideL*, const uint8_t * *pixelsR*, uint32_t *strideR*)

Add camera frame.

Parameters:

pObj	Pointer to SAC object.
pixelsL	Pointer to the pixels of luma channel, for the left camera.
strideL	Stride of the luma channel in bytes, for the left camera.
pixelsR	Pointer to the pixels of luma channel, for the right camera.
strideR	Stride of the luma channel in bytes, for the right camera.

void mvSAC_Deinitialize (mvSAC * *pObj*)

Deinitialize Stereo Auto-Calibration (SAC) object.

Parameters:

pObj	Pointer to SAC object.

MV_TRACKING_STATE mvSAC_GetCalibration (mvSAC * *pObj*, mvStereoConfiguration * *pStereoCfg*, mvSACStatus * *pStatus*)

Get the calibration result.

Parameters:

pObj	Pointer to SAC object.
pStereoCfg	Stereo configuration.
pStatus	SAC status. Set to nullptr if not needed.

Returns:

Tracking state. No calibration result is returned if state < 0.

mvSAC* mvSAC_Initialize (const mvCameraConfiguration * *pCfgL*, const mvCameraConfiguration * *pCfgR*, const float32_t *translation*[3], const mvSACConfiguration * *pSACCfg*)

Initialize Stereo Auto-Calibration (SAC) object.

Parameters:

pCfgL	Camera calibration parameters of the left camera. Parameters memoryStride and uvOffset are ignored.
pCfgR	Camera calibration parameters of the right camera. Parameters memoryStride and uvOffset are ignored.
translation	Translation from left camera's coordinate to right camera's coordinate, measured in meters.

pSACCfg	SAC configuration parameters.

Returns:

Pointer to SAC object; returns NULL if failed.

3.18 mvSRW.h File Reference

`#include <mv.h>`

3.18.1 Classes

- struct **mvImage**
- struct **mvFrame**
- struct **mvIMUData**
- struct **mvGPStimeSyncData**
- struct **mvGPSvelocityData**
- struct **mvAttitudeData**
- struct **mvCameraData**
- struct **mvCameraExtrinsicParameters**

3.18.2 Typedefs

- typedef struct **mvSRW_Writer mvSRW_Writer**
- typedef struct **mvSRW_Reader mvSRW_Reader**

3.18.3 Functions

- **mvSRW_Writer** * **mvSRW_Writer_Initialize** (const char *folderPath, mvMonoCameraInit *monoCam, mvStereoCameraInit *stereoCam)
- void **mvSRW_Writer_Deinitialize** (**mvSRW_Writer** *pObj)
- void **mvSRW_Writer_AddImage** (**mvSRW_Writer** *pObj, int64_t time, const uint8_t *pxls)
- void **mvSRW_Writer_AddStereoImage** (**mvSRW_Writer** *pObj, int64_t time, const uint8_t *pxlsL, const uint8_t *pxlsR)
- void **mvSRW_Writer_AddAccel** (**mvSRW_Writer** *pObj, int64_t time, float64_t x, float64_t y, float64_t z)
- void **mvSRW_Writer_AddGyro** (**mvSRW_Writer** *pObj, int64_t time, float64_t x, float64_t y, float64_t z)
- void **mvSRW_Writer_AddGpsTimeSync** (**mvSRW_Writer** *pObj, int64_t time, int64_t bias, int64_t drift, int64_t GPStimeUncertaintyStd)
- void **mvSRW_Writer_AddGpsVelocity** (**mvSRW_Writer** *pObj, int64_t time, float64_t x, float64_t y, float64_t z, float64_t xStd, float64_t yStd, float64_t zStd, uint16_t solutionInfo)
- void **mvSRW_Writer_AddCameraSettings** (**mvSRW_Writer** *pObj, int64_t time, float64_t gain, float64_t exposure, float64_t exposureScaled)
- void **mvSRW_Writer_AddAttitude** (**mvSRW_Writer** *pObj, **mvAttitudeData** *mvAttitudeDataPtr, int32_t numAttitudes)
- void **mvSRW_Writer_AddCameraParameters** (**mvSRW_Writer** *pObj, const char *name, **mvCameraConfiguration** *config)
- **mvSRW_Reader** * **mvSRW_Reader_Initialize** (const char *configDir)
- void **mvSRW_Reader_Deinitialize** (**mvSRW_Reader** *pObj)
- int **mvSRW_Reader_GetNumberOfCameras** (**mvSRW_Reader** *pObj)
- void **mvSRW_Reader_GetCameras** (**mvSRW_Reader** *pObj, mvCameraDescriptor *cameras)
- bool **mvSRW_Reader_GetCameraParameters** (**mvSRW_Reader** *pObj, const char *name, **mvCameraConfiguration** *camera)

- **mvFrame** * **mvSRW_Reader_GetNextFrame** (**mvSRW_Reader** *pObj)
- void **mvSRW_Reader_ReleaseFrame** (**mvSRW_Reader** *pObj, **mvFrame** *frame)
- **mvIMUData** * **mvSRW_Reader_GetNextGyro** (**mvSRW_Reader** *pObj, int64_t maxTimestamp)
- **mvIMUData** * **mvSRW_Reader_GetNextAccel** (**mvSRW_Reader** *pObj, int64_t maxTimestamp)
- void **mvSRW_Reader_ReleaseIMUData** (**mvSRW_Reader** *pObj, **mvIMUData** *imu)
- **mvGPStimeSyncData** * **mvSRW_Reader_GetNextGPStimeSync** (**mvSRW_Reader** *obj, int64_t maxTimestamp)
- void **mvSRW_Reader_ReleaseGPStimeSyncData** (**mvSRW_Reader** *pObj, **mvGPStimeSyncData** *timeSyncData)
- **mvGPSvelocityData** * **mvSRW_Reader_GetNextGPSvelocity** (**mvSRW_Reader** *pObj, int64_t maxTimestamp)
- void **mvSRW_Reader_ReleaseGPSvelocityData** (**mvSRW_Reader** *pObj, **mvGPSvelocityData** *velocityData)
- **mvAttitudeData** * **mvSRW_Reader_GetNextAttitude** (**mvSRW_Reader** *pObj, int64_t maxTimestamp)
- void **mvSRW_Reader_ReleaseAttitudeData** (**mvSRW_Reader** *pObj, **mvAttitudeData** *attitude)
- **mvStereoConfiguration** * **mvSRW_ReadStereoCalibrationFromXMLFile** (const char *fileName)
- bool **mvSRW_WriteStereoCalibrationToXML** (const char *filename, **mvStereoConfiguration** *stereoConfig)
- bool **mvSRW_WriteCameraExtrinsicParameters** (const char *filename, const **mvCameraExtrinsicParameters** *params)
- bool **mvSRW_ReadCameraExtrinsicParameters** (const char *filename, **mvCameraExtrinsicParameters** *params)

3.18.4 Detailed Description

mvSRW.h

Machine Vision, Sequence Reader Writer (SRW)

3.19 Overview

The SRW feature is for reading and writing data sequences that can be inputs into other MV features. One work flow might be to capture several cameras and IMU data using mvCapture which will write out a SRW sequence. That sequence can then be fed into a MV playback tool (e.g., mvDFSPlayback).

The sequences are saved as a directory structure of files. The directory structure needs to be the following:

```
data/
  accelerometer.xml
  attitude.xml
  cameraSettings.xml
  gyroscope.xml
  Configuration.VIO.playback.XML
data/Camera
  frame_00000.pgm
  . . .
  MetaInfo.xml
```

This directory and the contents is created by the Writer but the xml file describing the data (e.g., Configuration.VIO.playback.XML in this case) can be corrupted. It can be created by a user and placed in the data directory by hand.

The example config file looks like the following:

```
<?xml version='1.0' encoding='utf-8'?>
<Configuration>
  <Offline>
     <Camera folder="./Camera/" framerate="WAIT" loop="false" />
      <Sensor folder="./" loop="false" />
   </Offline>
</Configuration>
```

3.20 Limitations

The following list are some of the known limitations:

- Writer object must be properly de-initialized for file writing to complete.
- All data except images must fit into application RAM. However, if data is written faster than the disk write speed then all data including images must fit into memory.

3.20.1 Typedef Documentation

typedef struct mvSRW_Reader mvSRW_Reader

Sequence Reader for IMU and camera data.

typedef struct mvSRW_Writer mvSRW_Writer

Sequence Writer for IMU and camera data.

3.20.2 Function Documentation

bool mvSRW_ReadCameraExtrinsicParameters (const char * *filename*, mvCameraExtrinsicParameters * *params*)

Reads camera extrinsic parameters from XML file.

Parameters:

filename	Path to the xml file.

Returns:

Pointer to **mvCameraExtrinsicParameters** object.

void mvSRW_Reader_Deinitialize (mvSRW_Reader * *pObj*)

Deinitialize SequenceReader object.

Parameters:

pObj	Pointer to SequenceReader object.

bool mvSRW_Reader_GetCameraParameters (mvSRW_Reader * *pObj*, const char * *name*, mvCameraConfiguration * *camera*)

Read camera parameters from file for camera with corresponding name.

Parameters:

pObj	Pointer to SequenceReader object.
name	Name of camera to use a id.
cameras	Pre allocated memory for camera Configuration values.

void mvSRW_Reader_GetCameras (mvSRW_Reader * *pObj*, mvCameraDescriptor * *cameras*)

Get the descriptors of the camera.

Parameters:

pObj	Pointer to SequenceReader object.
cameras	Pre allocated memory for camera descriptors of available cameras.

mvIMUData* mvSRW_Reader_GetNextAccel (mvSRW_Reader * *pObj*, int64_t *maxTimestamp*)

Returns the next accelerometer reading.

Parameters:

pObj	Pointer to SequenceReader object.
maxTimestamp	Read accelerometer readings up to but not exceeding given timestamp.

Returns:

IMU data object that must be released after use.

mvAttitudeData* mvSRW_Reader_GetNextAttitude (mvSRW_Reader * *pObj*, int64_t *maxTimestamp*)

Returns the next attitude reading.

Parameters:

pObj	Pointer to SequenceReader object.
maxTimestamp	Read attitude readings up to but not exceeding given timestamp.

Returns:

Attitude data object that must be released after use.

mvFrame* mvSRW_Reader_GetNextFrame (mvSRW_Reader * *pObj*)

Reads and returns the next frame (image + time) [1 image for monocular and 2 images for stereo].

Parameters:

pObj	Pointer to SequenceReader object.

Returns:

Newly allocated frame object that must be released after use.

mvGPStimeSyncData* mvSRW_Reader_GetNextGPStimeSync (mvSRW_Reader * *obj*, int64_t *maxTimestamp*)

Returns the next gyro reading.

Parameters:

obj	Pointer to SequenceReader object.
maxTimestamp	Read GPS time sync readings up to but not exceeding given timestamp.

Returns:

GPS time sync data object that must be released after use.

mvGPSvelocityData* mvSRW_Reader_GetNextGPSvelocity (mvSRW_Reader * *pObj*, int64_t *maxTimestamp*)

Returns the next gyro reading.

Parameters:

pObj	Pointer to SequenceReader object.
maxTimestamp	Read GPS time sync readings up to but not exceeding given timestamp.

Returns:

GPS time sync data object that must be released after use.

mvIMUData* mvSRW_Reader_GetNextGyro (mvSRW_Reader * *pObj*, int64_t *maxTimestamp*)

Returns the next gyro reading.

Parameters:

pObj	Pointer to SequenceReader object.
maxTimestamp	Read gyro readings up to but not exceeding given timestamp.

Returns:

IMU data object that must be released after use.

int mvSRW_Reader_GetNumberOfCameras (mvSRW_Reader * *pObj*)

Get Number of Camera that the Reader found in Configuration (can be stereo and mono).

Parameters:

pObj	Pointer to SequenceReader object.

Returns:

Number of cameras.

mvSRW_Reader* mvSRW_Reader_Initialize (const char * *configDir*)

Initialize SequenceReader object.

Parameters:

folderPath	Location on storage where to save the sequence files.
width	Pixel Width of camera images.
height	Pixel Height of camera images.

Returns:

Pointer to SequenceWriter object; returns NULL if failed.

void mvSRW_Reader_ReleaseAttitudeData (mvSRW_Reader * *pObj*, mvAttitudeData * *attitude*)

Release IMU data memory after use.

Parameters:

pObj	Pointer to SequenceReader object.

void mvSRW_Reader_ReleaseFrame (mvSRW_Reader * *pObj*, mvFrame * *frame*)

Release frame data memory after use.

Parameters:

pObj	Pointer to SequenceReader object.

void mvSRW_Reader_ReleaseGPStimeSyncData (mvSRW_Reader * *pObj*, mvGPStimeSyncData * *timeSyncData*)

Release GPS time sync data memory after use.

Parameters:

pObj	Pointer to SequenceReader object.

void mvSRW_Reader_ReleaseGPSvelocityData (mvSRW_Reader * *pObj*, mvGPSvelocityData * *velocityData*)

Release GPS velocity data memory after use.

Parameters:

pObj	Pointer to SequenceReader object.

void mvSRW_Reader_ReleaseIMUData (mvSRW_Reader * *pObj*, mvIMUData * *imu*)

Release IMU data memory after use.

Parameters:

pObj	Pointer to SequenceReader object.

mvStereoConfiguration* mvSRW_ReadStereoCalibrationFromXMLFile (const char * *fileName*)

Reads MV standard XML Stereo Calibration file.

Parameters:

filename	Path to the calibration xml file.

Returns:

Pointer to **mvStereoConfiguration** object. Caller is responsible for deallocation using delete if XML file is ill formed the function returns null.

bool mvSRW_WriteCameraExtrinsicParameters (const char * *filename*, const mvCameraExtrinsicParameters * *params*)

Writes camera extrinsic parameters to XML file.

Parameters:

filename	Path to the xml file.

Returns:

Pointer to **mvCameraExtrinsicParameters** object.

void mvSRW_Writer_AddAccel (mvSRW_Writer * *pObj*, int64_t *time*, float64_t *x*, float64_t *y*, float64_t *z*)

Pass Accelerometer data to the SequenceWriter object.

Parameters:

pObj	Pointer to SequenceWriter object.
time	Timestamp of accelerometer data.
x	Accelerometer data for X axis.
y	Accelerometer data for Y axis.
z	Accelerometer data for Z axis.

void mvSRW_Writer_AddAttitude (mvSRW_Writer * *pObj*, mvAttitudeData * *mvAttitudeDataPtr*, int32_t *numAttitudes*)

Pass Attitude data to the SequenceWriter object.

Parameters:

pObj	Pointer to SequenceWriter object.
time	Pointer to the **mvAttitudeData** array.
numAttitudes	Size for the above array.

void mvSRW_Writer_AddCameraParameters (mvSRW_Writer * *pObj*, const char * *name*, mvCameraConfiguration * *config*)

Write file with name <name>.cal with camera parameters.

Parameters:

pObj	Pointer to SequenceWriter object.
name	Camera name, used for filename and should be same as in initialization.

config	Camera parameters to be written.

void mvSRW_Writer_AddCameraSettings (mvSRW_Writer * *pObj*, int64_t *time*, float64_t *gain*, float64_t *exposure*, float64_t *exposureScaled*)

Pass CameraSettings data to the SequenceWriter object.

Parameters:

pObj	Pointer to SequenceWriter object.
time	Timestamp of CameraSettings data.
gain	Gain settings applied to the camera.
exposure	Exposure time applied to the camera.

void mvSRW_Writer_AddGpsTimeSync (mvSRW_Writer * *pObj*, int64_t *time*, int64_t *bias*, int64_t *drift*, int64_t *GPStimeUncertaintyStd*)

Pass GPS time sync data to the SequenceWriter object.

Parameters:

pObj	Pointer to SequenceWriter object.
time	Timestamp of data in system time in nanoseconds.
bias	Time bias/offset (time bias/offset = GPS time - system time) in nanoseconds.
drift	Drift of system time w.r.t. GPS time (not currently used).
GPStimeUncertaintyStd	GPS time estimation uncertainty (set to -1 if not available).

void mvSRW_Writer_AddGpsVelocity (mvSRW_Writer * *pObj*, int64_t *time*, float64_t *x*, float64_t *y*, float64_t *z*, float64_t *xStd*, float64_t *yStd*, float64_t *zStd*, uint16_t *solutionInfo*)

Pass GPS velocity data to the SequenceWriter object.

Parameters:

pObj	Pointer to SequenceWriter object.
time	Timestamp of data in GPS time in nanoseconds.
x	GPS velocity in East direction in m/s.
y	GPS velocity in North direction in m/s.
z	GPS velocity in Up direction in m/s.
xStd	Standard deviation of velocity uncertainty in East in m/s.
yStd	Standard deviation of velocity uncertainty in North in m/s.
zStd	Standard deviation of velocity uncertainty in Up in m/s.

solutionInfo	Fix type/quality: the last 3 bits being '100' represents a good message (if available, otherwise set to 4).

void mvSRW_Writer_AddGyro (mvSRW_Writer * *pObj*, int64_t *time*, float64_t *x*, float64_t *y*, float64_t *z*)

Pass Gyroscope data to the SequenceWriter object.

Parameters:

pObj	Pointer to SequenceWriter object.
time	Timestamp of Gyro data.
x	Gyro data for X axis.
y	Gyro data for Y axis.
z	Gyro data for Z axis.

void mvSRW_Writer_AddImage (mvSRW_Writer * *pObj*, int64_t *time*, const uint8_t * *pxls*)

Pass camera frame to the MV SequenceWriter object.

Parameters:

pObj	Pointer to SequenceWriter object.
time	Timestamp of camera frame.
pxls	Pointer to camera frame data.

void mvSRW_Writer_AddStereoImage (mvSRW_Writer * *pObj*, int64_t *time*, const uint8_t * *pxlsL*, const uint8_t * *pxlsR*)

Pass stereo camera frame to the MV SequenceWriter object.

Parameters:

pObj	Pointer to SequenceWriter object.
time	Timestamp of camera frame.
pxlsL	Pointer to left camera frame data.
pxlsR	Pointer to right camera frame data.

void mvSRW_Writer_Deinitialize (mvSRW_Writer * *pObj*)

Deinitialize SequenceWriter object.

Parameters:

pObj	Pointer to SequenceWriter object.

mvSRW_Writer* mvSRW_Writer_Initialize (const char * *folderPath*, mvMonoCameraInit * *monoCam*, mvStereoCameraInit * *stereoCam*)

Initialize SequenceWriter object.

Parameters:

folderPath	Location on storage where to save the sequence files.
monoCam	Pointer to monocular camera object.
stereoCam	Pointer to stereo camera object.

Returns:

Pointer to SequenceWriter object; returns NULL if failed.

bool mvSRW_WriteStereoCalibrationToXML (const char * *filename*, mvStereoConfiguration * *stereoConfig*)

Writes Stereo configuration into MV standard XML format.

Parameters:

filename	Path to filename.
stereoConfig	Stereo configuration to writer.

Returns:

true on success false otherwise.

3.21 mvVISLAM.h File Reference

`#include <mv.h>`

3.21.1 Classes

- struct **mvVISLAMPose**
- struct **mvVISLAMMapPoint**

3.21.2 Typedefs

- typedef class **mvVISLAM mvVISLAM**

3.21.3 Functions

- **mvVISLAM** * **mvVISLAM_Initialize** (const **mvCameraConfiguration** *camera, const float32_t readoutTime, const float32_t *tbc, const float32_t *ombc, const float32_t delta, const float32_t *std0Tbc, const float32_t *std0Ombc, const float32_t std0Delta, const float32_t accelMeasRange, const float32_t gyroMeasRange, const float32_t stdAccelMeasNoise, const float32_t stdGyroMeasNoise, const float32_t stdCamNoise, const float32_t minStdPixelNoise, const float32_t failHighPixelNoiseScaleFactor, const float32_t logDepthBootstrap, const bool useLogCameraHeight, const float32_t logCameraHeightBootstrap, const bool noInitWhenMoving, const float32_t limitedIMUbWtrigger, const char *staticMaskFileName, const float32_t gpsImuTimeAlignment, const float32_t *tba)
- void **mvVISLAM_Deinitialize** (**mvVISLAM** *pObj)
- void **mvVISLAM_AddImage** (**mvVISLAM** *pObj, int64_t time, const uint8_t *pxls)
- void **mvVISLAM_AddAccel** (**mvVISLAM** *pObj, int64_t time, float64_t x, float64_t y, float64_t z)
- void **mvVISLAM_AddGyro** (**mvVISLAM** *pObj, int64_t time, float64_t x, float64_t y, float64_t z)

- void **mvVISLAM_AddGPSvelocity** (**mvVISLAM** *pObj, int64_t time, float64_t velocityEast, float64_t velocityNorth, float64_t velocityUP, float64_t measCovVelocity[3][3], uint16_t solutionInfo)
- void **mvVISLAM_AddGPStimeSync** (**mvVISLAM** *pObj, int64_t time, int64_t bias, int64_t gpsTimeStdDev)
- const **mvVISLAMPose mvVISLAM_GetPose** (**mvVISLAM** *pObj)
- int **mvVISLAM_HasUpdatedPointCloud** (**mvVISLAM** *pObj)
- int **mvVISLAM_GetPointCloud** (**mvVISLAM** *pObj, **mvVISLAMMapPoint** *pPoints, uint32_t maxPoints)
- void **mvVISLAM_Reset** (**mvVISLAM** *pObj, bool resetPose)

3.21.4 Detailed Description

mvVISLAM.h

Machine Vision, Visual-Inertial Simultaneous Fusion Localization And Mapping (VISLAM)

3.22 Overview

VISLAM provides 6-DOF localization and pose estimation for various applications. It has been tuned for robot use cases in particular.

In addition to the initialization parameters, there are other things to consider when attempting to get the best possible performance out of VISLAM. A good camera calibration performed specifically for a given camera has the potential to significantly reduce the overall odometry drift rather than the default calibration provided in examples.

Furthermore, rich motion just after VISLAM starts can accelerate the state space convergence and lead to lower drift. For the drone application, rolling/pitching and high linear accelerations are good types of motion for better convergence.

3.23 Limitations

The following list are some of the known limitations:

- The state may drift before takeoff if the IMU cutoff frequency is set to below the frequency of vibration sources on the board (fan, propellers).
- Landing in a scenario where the closest features are far away may not sufficiently constrain the position estimate, causing the drone to drift on the ground if GPS velocity estimates are not provided
- Flying over water violates the assumption of feature stationary; system will reset if GPS velocity estimates are not provided
- Flying at high altitudes drives up velocity uncertainty which can cause problems if all points (for which depth has converged) are lost after excessive yawing and GPS velocity estimates are not available.

3.23.1 Typedef Documentation

typedef class mvVISLAM mvVISLAM

Visual-Inertial Simultaneous Fusion Localization and Mapping (VISLAM)

3.23.2 Function Documentation

void mvVISLAM_AddAccel (mvVISLAM * *pObj*, int64_t *time*, float64_t *x*, float64_t *y*, float64_t *z*)

Pass Accelerometer data to the VISLAM object.

Parameters:

pObj	Pointer to VISLAM object.
time	Timestamp of data in nanoseconds in system time.
x	Accelerometer data for X axis in m/s^2.
y	Accelerometer data for Y axis in m/s^2.
z	Accelerometer data for Z axis in m/s^2.

void mvVISLAM_AddGPStimeSync (mvVISLAM * *pObj*, int64_t *time*, int64_t *bias*, int64_t *gpsTimeStdDev*)

Pass GPS time bias data to the VISLAM object.

Parameters:

pObj	Pointer to VISLAM object.
time	Timestamp of data in system time in nanoseconds.
bias	Time bias/offset (time bias/offset = GPS time - system time) in nanoseconds.
gpsTimeStdDev	GPS time uncertainty (if available, otherwise set to -1).

void mvVISLAM_AddGPSvelocity (mvVISLAM * *pObj*, int64_t *time*, float64_t *velocityEast*, float64_t *velocityNorth*, float64_t *velocityUP*, float64_t *measCovVelocity*[3][3], uint16_t *solutionInfo*)

Pass GPS velocity data to the VISLAM object

Parameters:

pObj	Pointer to VISLAM object
time	Timestamp of data in GPS time in nanoseconds
velocityEast	GPS velocity data in East direction in m/s.
velocityNorth	GPS velocity data in North direction in m/s.
velocityUP	GPS velocity data in Up direction in m/s.
measCovVelocity	GPS velocity measurement error co-variance, fields in (m/s)^2.
solutionInfo	Fix type/quality: the last 3 bits being '100' represents a good message (if available, otherwise set to 4).

void mvVISLAM_AddGyro (mvVISLAM * *pObj*, int64_t *time*, float64_t *x*, float64_t *y*, float64_t *z*)

Pass Gyroscope data to the VISLAM object.

Parameters:

pObj	Pointer to VISLAM object.
time	Timestamp of data in nanoseconds in system time.
x	Gyro data for X axis in rad/s.
y	Gyro data for Y axis in rad/s.
z	Gyro data for Z axis in rad/s.

void mvVISLAM_AddImage (mvVISLAM * *pObj*, int64_t *time*, const uint8_t * *pxls*)

Add the camera frame to the VISLAM object and trigger processing (a frame update) on the newly added image while utilizing any already added IMU samples, including timestamps occurring after the image, to fully propagate the pose forward to the most recent IMU sample.

NOTE: All other sensor data occurring before this image must be added first before calling this function otherwise that older data will be dropped at the next call of this function.

Parameters:

pObj	Pointer to VISLAM object.
time	Timestamp of camera frame in nanoseconds in system time. Time must be center of exposure time, not start of frame or end of frame.
pxls	Pointer to camera frame 8-bit grayscale luminance data (VGA).

void mvVISLAM_Deinitialize (mvVISLAM * *pObj*)

Deinitialize VISLAM object.

Parameters:

pObj	Pointer to VISLAM object.

int mvVISLAM_GetPointCloud (mvVISLAM * *pObj*, mvVISLAMMapPoint * *pPoints*, uint32_t *maxPoints*)

Grab point cloud.

Parameters:

pObj	Pointer to VISLAM object.
pPoints	Pre-allocated array of **mvVISLAMMapPoint** structure to be filled in by VISLAM with current map points.
maxPoints	Max number of points requested. Should match allocated size of pPoints.

Returns:

Number of points filled into the pPoints array.

const mvVISLAMPose mvVISLAM_GetPose (mvVISLAM * *pObj*)

Grab last computed pose.

Parameters:

pObj	Pointer to VISLAM object.

Returns:

Computed pose from previous frame and IMU data.

int mvVISLAM_HasUpdatedPointCloud (mvVISLAM * *pObj*)

Inquire if VISLAM has new map points.

Parameters:

pObj	Pointer to VISLAM object.

Returns:

Number of map points currently being observed and estimated.

mvVISLAM* mvVISLAM_Initialize (const mvCameraConfiguration * *camera*, const float32_t *readoutTime*, const float32_t * *tbc*, const float32_t * *ombc*, const float32_t *delta*, const float32_t * *std0Tbc*, const float32_t * *std0Ombc*, const float32_t *std0Delta*, const float32_t *accelMeasRange*, const float32_t *gyroMeasRange*, const float32_t *stdAccelMeasNoise*, const float32_t *stdGyroMeasNoise*, const float32_t *stdCamNoise*, const float32_t *minStdPixelNoise*, const float32_t *failHighPixelNoiseScaleFactor*, const float32_t *logDepthBootstrap*, const bool *useLogCameraHeight*, const float32_t *logCameraHeightBootstrap*, const bool *noInitWhenMoving*, const float32_t *limitedIMUbWtrigger*, const char * *staticMaskFileName*, const float32_t *gpsImuTimeAlignment*, const float32_t * *tba*)

Initialize VISLAM object. A few parameters may significantly impact the performance of the VISLAM algorithm. Some parameters affect the initial convergence of VISLAM, which impacts the overall drift in the estimated pose. The following parameters should have particular attention paid to them: logDepthBootstrap, useLogCameraHeight, logCameraHeightBootstrap, and limitedIMUbWtrigger.

Parameters:

camera	Pointer to camera intrinsic calibration parameters.
readoutTime	Frame readout time (seconds). n times row readout time. Set to 0 for global shutter camera. Frame readout time should be (close to) but smaller than the rolling shutter camera frame period.
tbc	Pointer to accelerometer-camera translation misalignment vector (meters). T_{bc} is the translation of the origin of the camera (c) frame relative to that of

	the body (b) or accelerometer frame in the body frame. T_{bc} setting can be verified by checking estimated T_{bc}.
ombc	Pointer to accelerometer-camera misalignment vector (radians). {bc} is the corresponding rotation in exponential coordinates. Can be used together with T_{bc} to rotate vector in camera frame x_c to IMU frame x_imu via [R|T]x_c = x_imu. {bc} settings can be verified by checking estimated R_{bc} mapped to exponential coordinates.
delta	Camera-inertial timestamp misalignment (seconds). Ideally this is within about 1 ms of the true value. Delta can be verified by checking the estimated time alignment.
std0Tbc	Pointer to initial uncertainty in accelerometer-camera translation vector (meters).
std0Ombc	Pointer to initial uncertainty in accelerometer-camera orientation vector (rad.).
std0Delta	Initial uncertainty in time misalignment estimate (seconds).
accelMeasRange	Accelerometer sensor measurement range (m/s^2).
gyroMeasRange	Gyro sensor measurement range (rad./s).
stdAccelMeasNoise	Standard deviation of accelerometer measurement noise (m/s^2).
stdGyroMeasNoise	Standard deviation of gyro measurement noise (rad./s).
stdCamNoise	Standard dev of camera noise per pixel.
minStdPixelNoise	Minimum of standard deviation of feature measurement noise in pixels.
failHighPixelNoiseScaleFactor	Scales measurement noise and compares against search area (is search area large enough to reliably compute measurement noise covariance matrix).
logDepthBootstrap	Initial point depth [log(meters)], where log is the natural log. By default, initial depth is set to 1m. However, if e.g. a downward facing camera on a drone is used and it can be assumed that feature depth at initialization is always e.g. 4cm, then we can set this parameter to 4cm (or -3.2). This will improve tracking during takeoff, accelerate state space convergence, and lead to more accurate and robust pose estimates.
useLogCameraHeight	Use logCameraHeightBootstrap instead of logDepthBootstrap.

logCameraHeightBootstrap	Initializes point depth based on known geometry, assumes (1) camera pointing partially at ground plane and (2) board/IMU aligned with gravity at start (= accelerometer measures roughly [0, 0, -9.8] in units of m/s^2), required input is camera height over ground (log(meters)), log is natural log. Understanding when to use logDepthBootstrap versus logCameraHeightBootstrap and how to set these values appropriately can improve the initialization of VISLAM and has the potential to reduce the amount of odometry drift observed.
noInitWhenMoving	Set if device is stationary w.r.t. surface when initializing (e.g. drone) based on camera, not on IMU: supports device on moving surface.
limitedIMUbWtrigger	To prevent tracking failure during/right after (hard) landing: If sum of 3 consecutive accelerometer samples in any dimension divided by 4.3 exceed this threshold, IMU measurement noise is increased (and resets become more likely); **NOTE:** if platform vibrates heavily during flight, this may trigger mid- flight; if poseQuality in **mvVISLAMPose** drops to MV_TRACKING_STATE_LOW_QUALITY during flight, improve mechanical dampening (and/or increase threshold) RECOMMEND: 150m/s^2 / 4.3 ~= 35
staticMaskFileName	1/4 resolution image (w.r.t. VGA), 160x120, PGM format, the part of the camera view for which pixels are set to 255 is blocked from feature detection useful, e.g., to avoid detecting & tracking points on landing gear reaching into camera view.
gpsImuTimeAlignment	GPS-inertial timestamp misalignment (seconds), negative if GPS time stamping is delayed relative to IMU time stamping, ideally this is within about 1 ms of the true value.
tba	Pointer to accelerometer-GPS antenna translation misalignment vector/lever arm (meters). T_{ba} is the translation of the origin of the GPS antenna (a) frame relative to that of the body (b) or accelerometer frame in the body frame.

Returns:

Pointer to VISLAM object; returns NULL if failed.

void mvVISLAM_Reset (mvVISLAM * *pObj*, bool *resetPose*)

Resets the EKF from an external source. EKF will try to reinitialize in the subsequent camera frame. To properly initialize after reset, device should not be rotating, moving a lot, camera look at 10+ features.

Parameters:

pObj	Pointer to VISLAM object.
resetPose	false: initializes with last good pose after triggering reset true: initializes with "zero" pose after triggering reset

---

3.24 mvVM.h File Reference

```
#include <mv.h>
#include <string.h>
```

3.24.1 Classes

- struct **mvVM_IntegrationConfiguration**
- struct **mvVM_CollisionInfo**

3.24.2 Typedefs

- typedef struct **mvVM mvVM**

3.24.3 Functions

- **mvVM** * **mvVM_Initialize** (const float32_t sampleDistance[3])
- void **mvVM_Deinitialize** (**mvVM** *map)
- void **mvVM_GetSampleDistance** (**mvVM** *map, float32_t sampleDistance[3])
- void **mvVM_MoveOriginTo** (**mvVM** *map, float32_t origin[3])
- void **mvVM_Clear** (**mvVM** *map)
- void **mvVM_IntegrateDepthMap** (**mvVM** *map, const float32_t *data, const **mvCameraConfiguration** *camera, const **mvPose6DRT** *registration, const **mvVM_IntegrationConfiguration** *config)
- void **mvVM_IntegrateDepthMapUInt16** (**mvVM** *map, const uint16_t *data, const **mvCameraConfiguration** *camera, const **mvPose6DRT** *registration, const **mvVM_IntegrationConfiguration** *config)
- **MV_COLLISION mvVM_CheckCollisionWithPoint** (const **mvVM** *map, const float32_t A[3], const float32_t threshold, **mvVM_CollisionInfo** *info)
- **MV_COLLISION mvVM_CheckCollisionWithBox** (const **mvVM** *map, const float32_t lower[3], const float32_t upper[3], const float32_t threshold, **mvVM_CollisionInfo** *info)
- **MV_COLLISION mvVM_CheckCollisionWithLine** (const **mvVM** *map, const float32_t A[3], const float32_t B[3], const float32_t threshold, **mvVM_CollisionInfo** *info)
- **MV_COLLISION mvVM_GetMinimalDistanceToPoint** (const **mvVM** *map, const float32_t A[3], const float32_t maximalDistance, const float32_t threshold, float32_t *distance, **mvVM_CollisionInfo** *info)
- **MV_COLLISION mvVM_GetMinimalDistanceToBox** (const **mvVM** *map, const float32_t lower[3], const float32_t upper[3], const float32_t maximalDistance, const float32_t threshold, float32_t *distance, **mvVM_CollisionInfo** *info)
- void **mvVM_ClipAgainstBox** (**mvVM** *map, const float32_t lower[3], const float32_t upper[3])
- void **mvVM_ClipAgainstSphere** (**mvVM** *map, const float32_t center[3], const float32_t radius)
- void **mvVM_SetBoxFree** (**mvVM** *map, const float32_t lower[3], const float32_t upper[3])

- void **mvVM_SetBoxOccupied** (**mvVM** *map, const float32_t lower[3], const float32_t upper[3])
- void **mvVM_ExtractSamplePoints** (const **mvVM** *map, const float32_t threshold, float32_t *points, size_t *numberPoints)
- void **mvVM_ExtractSurfacePoints** (const **mvVM** *map, const float32_t threshold, float32_t *vertices, size_t *numberVertices)
- void **mvVM_ExtractSurfaceMesh** (const **mvVM** *map, const float32_t threshold, float32_t *vertices, size_t *numberVertices, uint32_t *indices, size_t *numberIndices)

3.24.4 Detailed Description

mvVM.h

Machine Vision SDK, Voxel Map (VM)

3.24.5 Typedef Documentation

typedef struct mvVM mvVM

Voxel Mapping (VM)

3.24.6 Function Documentation

MV_COLLISION mvVM_CheckCollisionWithBox (const mvVM * *map*, const float32_t *lower*[3], const float32_t *upper*[3], const float32_t *threshold*, mvVM_CollisionInfo * *info*)

Checks if an axis aligned box in space hits the map

Parameters:

map	VM object.
lower	Lower corner of the box.
upper	Upper corner of the box.
threshold	Map threshold value to tread a sample in the map as occupied.
info	Optional structure to return the sample point that collided and more information. If info == NULL, then it is ignored.

Returns:

MV_COLLISION value describing the result as no collision, collision, or unknown.

MV_COLLISION mvVM_CheckCollisionWithLine (const mvVM * *map*, const float32_t *A*[3], const float32_t *B*[3], const float32_t *threshold*, mvVM_CollisionInfo * *info*)

Checks if a line in space hits the map.

Parameters:

map	VM object.
A	Start point of the line.
B	End point of the line.
threshold	Map threshold value to tread a sample in the map as occupied.
info	Optional structure to return the sample point that collided and more information. If info == NULL, then it is ignored.

Returns:

MV_COLLISION value describing the result as no collision, collision, or unknown.

MV_COLLISION mvVM_CheckCollisionWithPoint (const mvVM * *map*, const float32_t *A*[3], const float32_t *threshold*, mvVM_CollisionInfo * *info*)

Checks if a point in space is occupied.

Parameters:

map	VM object.
A	Location of the point in 3D space in the world coordinate frame.
threshold	Map threshold value to tread a sample in the map as occupied.
info	Optional structure to return the sample point that collided and more information. If info == NULL, then it is ignored.

Returns:

MV_COLLISION value describing the result as no collision, collision, or unknown.

void mvVM_Clear (mvVM * *map*)

Clears the data in the map

Parameters:

map	VM object

void mvVM_ClipAgainstBox (mvVM * *map*, const float32_t *lower*[3], const float32_t *upper*[3])

Clips the map against a given axis aligned box. All data within the map outside of the box is deleted. Some fringe around the box can remain.

Parameters:

map	VM object.
lower	Lower corner of the box.

upper	Upper corner of the box.

void mvVM_ClipAgainstSphere (mvVM * *map*, const float32_t *center*[3], const float32_t *radius*)

Clips the map against a given sphere. All data within the map outside of the sphere is deleted. Some fringe around the sphere can remain.

Parameters:

map	VM object.
center	Center of the sphere.
radius	Radius of the sphere.

void mvVM_Deinitialize (mvVM * *map*)

Deinitialize VM object

Parameters:

map	VM object.

void mvVM_ExtractSamplePoints (const mvVM * *map*, const float32_t *threshold*, float32_t * *points*, size_t * *numberPoints*)

Extracts the occupied sample locations in the volume grid that have non-empty values. This can be used to create a representation of the occupied blocks - not the estimated surface.

Parameters:

map	VM object.
points	Pointer to a buffer of floats in {x0,y0,z0}, {x1,y1,z1}, ... format. Use NULL if wanting numberPoints first: size_t numberVertices = 0; mvVM_ExtractSamplePoints(map, 0, NULL, &numberVertices);
numberPoints	Pointer to size value of the number of points written to the buffer.

void mvVM_ExtractSurfaceMesh (const mvVM * *map*, const float32_t *threshold*, float32_t * *vertices*, size_t * *numberVertices*, uint32_t * *indices*, size_t * *numberIndices*)

Extracts a surface mesh.

Parameters:

map	VM object.
vertices	Pointer to a buffer of floats in {x0,y0,z0}, {x1,y1,z1}, ... format.
numberVertices	Pointer to size value of the number of points written to the buffer.
indices	Pointer to a buffer of triangle vertex indices of the mesh.
numberIndices	Pointer to size value of the number of indices written to the buffer.

void mvVM_ExtractSurfacePoints (const mvVM * *map*, const float32_t *threshold*, float32_t * *vertices*, size_t * *numberVertices*)

Extracts the vertex locations on the surface.

Parameters:

map	VM object.
vertices	Pointer to a buffer of floats in {x0,y0,z0}, {x1,y1,z1}, ... format. Use NULL if wanting numberPoints first: size_t numberVertices = 0; mvVM_ExtractSurfacePoints(map, 0, NULL, &numberVertices);
numberVertices	Pointer to size value of the number of points written to the buffer.

MV_COLLISION mvVM_GetMinimalDistanceToBox (const mvVM * *map*, const float32_t *lower*[3], const float32_t *upper*[3], const float32_t *maximalDistance*, const float32_t *threshold*, float32_t * *distance*, mvVM_CollisionInfo * *info*)

Returns the closest hit point on the map and the distance to a given axis aligned box. The returned point does not need to be the unique solution, there might be more points with the same distance.

Parameters:

map	VM object.
lower	Lower corner of the box.
upper	Upper corner of the box.
maximalDistance	Maximal distance to search for.
threshold	Map threshold value to tread a sample in the map as occupied.
distance	Pointer to return the distance found.
minimalPoint	Map sample location that was found to be closest.

MV_COLLISION mvVM_GetMinimalDistanceToPoint (const mvVM * *map*, const float32_t *A*[3], const float32_t *maximalDistance*, const float32_t *threshold*, float32_t * *distance*, mvVM_CollisionInfo * *info*)

Returns the closest hit point on the map and the distance to a given point. The returned point does not need to be the unique solution, there might be more points with the same distance.

Parameters:

map	VM object.
A	Point in space.
maximalDistance	Maximal distance to search for.
threshold	Map threshold value to tread a sample in the map as occupied.
distance	Pointer to return the distance found.
minimalPoint	Map sample location that was found to be closest.

void mvVM_GetSampleDistance (mvVM * *map*, float32_t *sampleDistance*[3])

Get the sample distances in meters for the map.

Parameters:

map	VM object.
sampleDistance	Contains the sample distances in meters along X, Y and Z axis.

mvVM* mvVM_Initialize (const float32_t *sampleDistance*[3])

Initialize a Voxel Map object.

Parameters:

sampleDistance	Distances in meters between samples along X, Y, and Z axis.

Returns:

On success pointer to VM object, NULL pointer on failure.

void mvVM_IntegrateDepthMap (mvVM * *map*, const float32_t * *data*, const mvCameraConfiguration * *camera*, const mvPose6DRT * *registration*, const mvVM_IntegrationConfiguration * *config*)

Integrates a new depth map into the map.

NOTE: Updates the volume map by integrating the depth map.

Parameters:

map	VM object.
data	Pointer to the raw depth map image data. The values are interpreted as depth measurements in the same units as the sample distances.
camera	Pointer to a camera calibration object. Non-linear distortion parameters are not supported, a linear camera model is assumed.
registration	Pointer to a pose object, storing the transformation from world coordinate system to camera coordinate system.
noiseModel	Single parameter noise model for the depth map, here the "ramp" around the measured depth values.
filterModel	Single parameter filter model for the map, here the maximal weight of the running average filter.

void mvVM_IntegrateDepthMapUInt16 (mvVM * *map*, const uint16_t * *data*, const mvCameraConfiguration * *camera*, const mvPose6DRT * *registration*, const mvVM_IntegrationConfiguration * *config*)

Integrates a new depth map into the map.

NOTE: Updates the volume map by integrating the depth map.

Parameters:

map	VM object.
data	Pointer to the raw depth map image data. The values are interpreted as depth measurements in the same units as the sample distances.
camera	Pointer to a camera calibration object. Non-linear distortion parameters are not supported, a linear camera model is assumed.
registration	Pointer to a pose object, storing the transformation from world coordinate system to camera coordinate system.
config	Pointer to integration configuration data.

void mvVM_MoveOriginTo (mvVM * *map*, float32_t *origin*[3])

Moves the origin of the map to a new origin.

Parameters:

map	VM object.
origin	New origin in meters and current world coordinates. The value origin is changed to reflect the actual new origin which may differ from the provided one, due to quantization of the map samples.

void mvVM_SetBoxFree (mvVM * *map*, const float32_t *lower*[3], const float32_t *upper*[3])

Sets samples in the volume covered by a box to free.

Parameters:

map	VM object.
lower	Lower corner of the box.
upper	Upper corner of the box.

void mvVM_SetBoxOccupied (mvVM * *map*, const float32_t *lower*[3], const float32_t *upper*[3])

Sets samples in the volume covered by a box to be occupied.

Parameters:

map	VM object.
lower	Lower corner of the box.
upper	Upper corner of the box.