IMMERVISION

CONFIDENTIAL

Immervision Enables 2.0

Image Processing

Programmer’s Guide

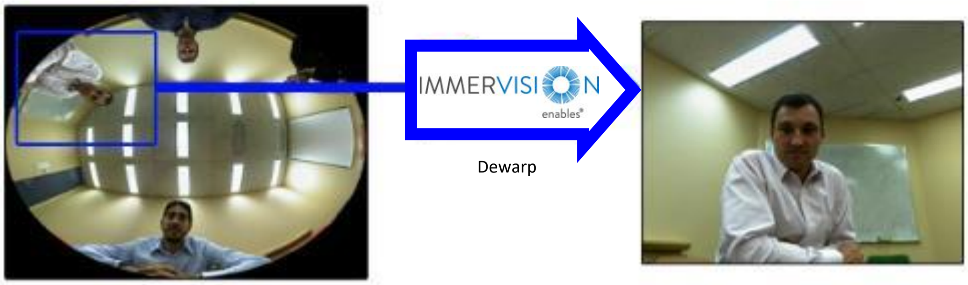
Document version: 2.12
Document release date: 2019-10-16

Contents

DESCRIPTION

Introduction

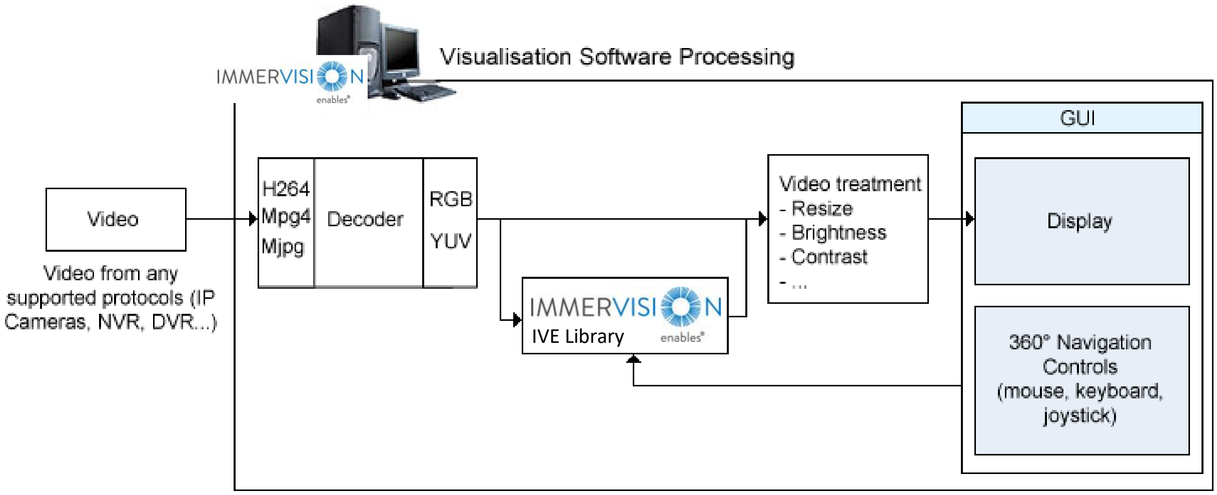
Immervision Enables 2.0 SDK is a tool allowing the developers to enable dewarping engine. The dewarping engine dewarps video frames captured with the certified panomorph lenses, and generates images or videos without distortions.



Dewarping of a Panomorph video

Integration Overview
The library was designed to be non-invasive. The library only needs to be integrated in the imaging software as a part of the video processing stream.

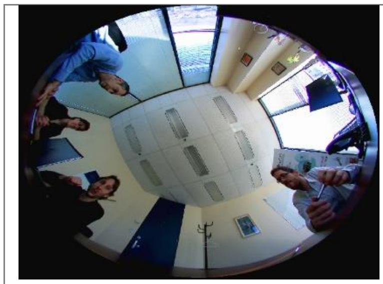
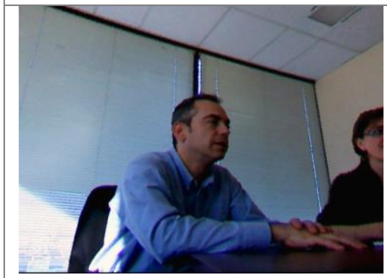
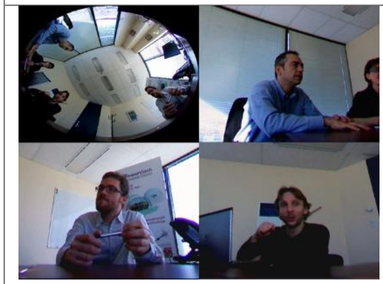
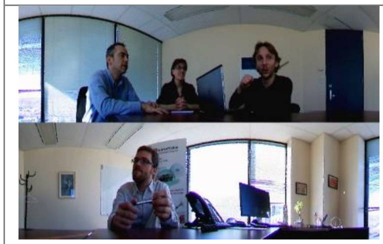
The visualization software displays videos from any standard Immervision Enables sources (e.g. protocols of IP, mobile phone, wearable, analog cameras, DVR, capture cards, etc.).
Like other standard video processes (e.g. resize, brightness, contrast, etc.), the Library processes uncompressed video frames coming from capture devices equipped with panomorph lenses to generate an uncompressed dewarped picture (undistorted view).

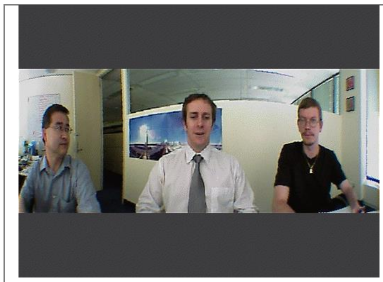
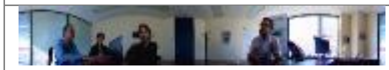
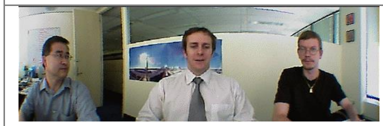
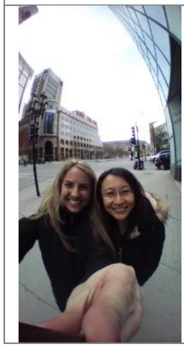

IMMERVISION
enables®
Visualisation Software Processing
Video
Video from any supported protocols (IP Cameras, NVR, DVR...)
H264
Mpg4
Mjpg
Decoder
RGB
YUV
IMMERVISION
IVE Library
enables®
Video treatment
- Resize
- Brightness
- Contrast
- ...
GUI
Display
360° Navigation Controls (mouse, keyboard, joystick)

Viewing Modes

The Immervision Enables library offers a choice of 4 viewing modes to display the video without distortion:

- Single mode
- Quad mode (4 single mode)
- Perimeter mode
- Vertical Selfie mode
- Customizable Perimeter mode

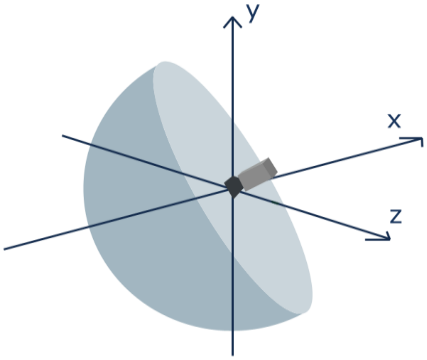
	Immervision Enables video frames Video frames from a certified panomorph lenses. Note: The image must contain the full lens footprint (circular or elliptic)
	Single mode Pan, Tilt & Zoom parameters are applied to move the virtual camera and navigate in 360°.
	Quad mode 4 virtual Single views are computed. The developer can apply pan, tilt & zoom parameters simultaneously.
	Perimeter mode When the capture device is placed vertically with the lens up or down, the library computes 2 strips of 180° field of view. This perimeter mode displays the full surround view around the capture device.

	Perimeter mode (Front Facing) When the capture device is placed horizontally, the software computes a strip of 180° field of view. This perimeter mode displays the scene in front of the capture device. The display ratio is fixed to 2:1.
	Customizable Perimeter mode When the capture device is placed vertically with the lens up or down, the software computes 1 strip of 360° field of view. This perimeter mode displays the full surround view around the capture device. The horizontal and vertical Field of View are customizable.
	Customizable Perimeter mode (Front Facing) When the capture device is placed horizontally, the software computes a strip of 180° field of view. This perimeter mode displays the scene in front of the capture device. The horizontal and vertical Field of View are customizable.
	Vertical Selfie mode The software computes a strip of 180° field of view from Bottom to Top. The display ratio is fixed to 2:1.

Capture device Position and Rotation

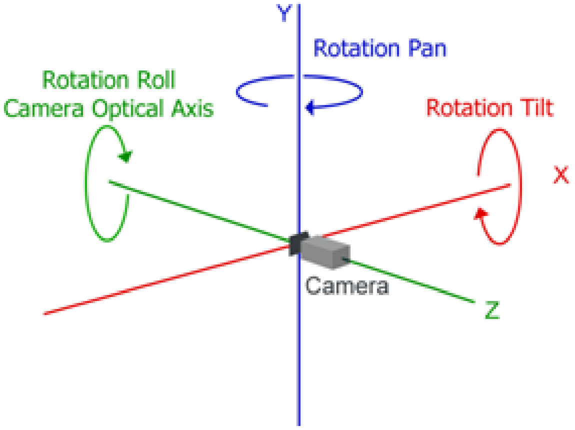
The Immervision Enables 2.0 library supports any capture device movements (rotations).

The capture device can rotate by any angles. The rotation angles are defined using the coordinates system described below.



Camera rotation referential

The capture device rotation is set using 3 angles: pan, tilt and roll in the coordinate system below.
(note this coordinate system is the same than the Wall coordinates system which is the default one)



The function SetRotation(pan, tilt, roll) allows the library to position the capture device correctly.

360° Navigation Referential

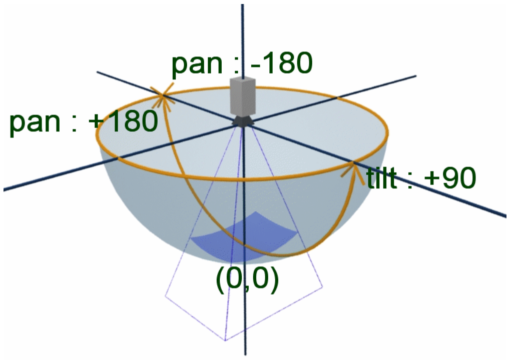
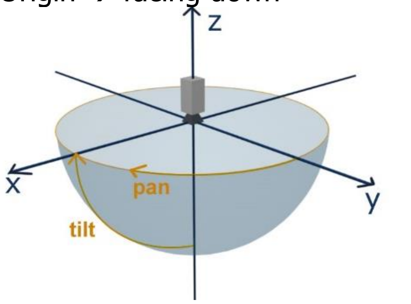
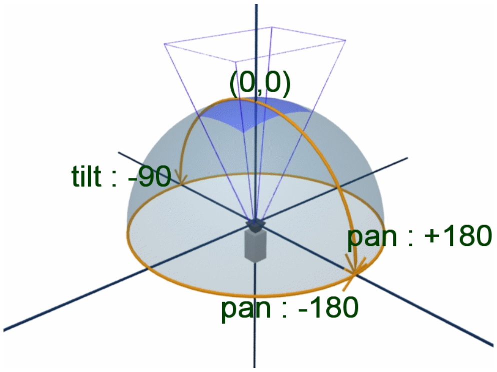
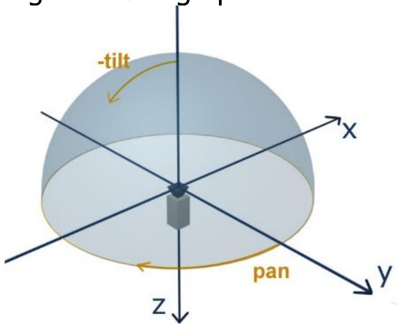
The library also offers a choice of three 360° navigation 3 referentials:

- Wall position
- Ceiling position
- Ground position

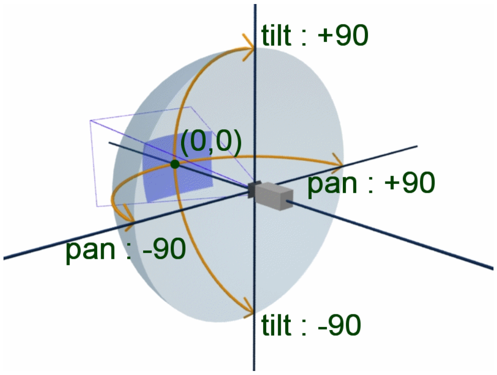
Each of these referential:

- Defines the origin and the orientation of the 360° navigation referential. This affects the virtual camera movement
- Presets the capture device rotation values.

SINGLE OR QUAD MODES

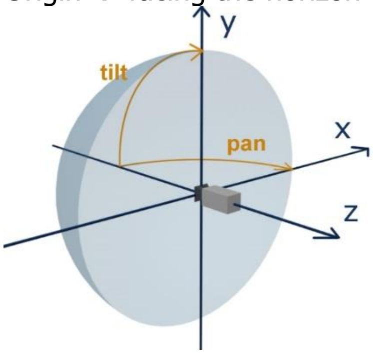
Pan: -180° to +180° Tilt: 0° to 90° (1) 	Ceiling position **360° navigation referential** Origin → facing down  **Capture device rotation** pan=0, tilt=-90, roll=0
Pan: -180° to +180° Tilt: -90° to 0° (1) 	Ground position **360° navigation referential** Origin → facing up  **Capture device rotation** pan=0, tilt=90, roll=0

Pan: -90° to +90°[1]
Tilt: -90° to 90°[1]



Wall position

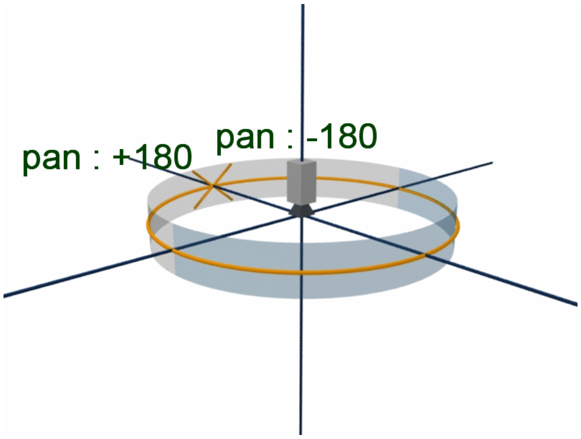
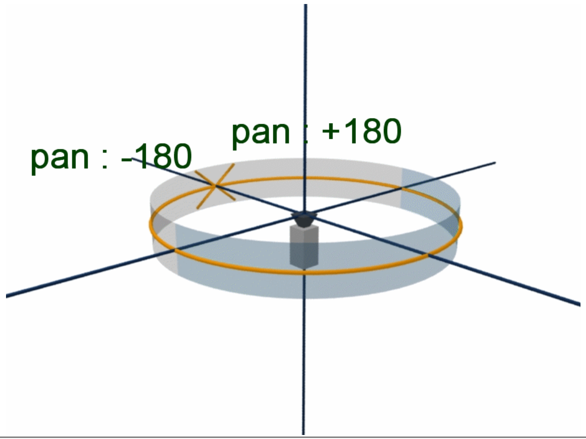
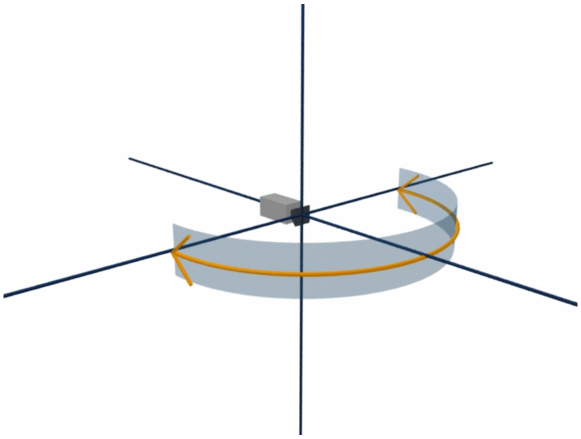
360° navigation referential
Origin → facing the horizon



Capture device rotation
pan=0, tilt=0, roll=0

[1] Default values depend on the real Field Of View of the panomorph lens. If the lens Field of View is 190°, the pan will be -95° to +95°.

PERIMETER MODE

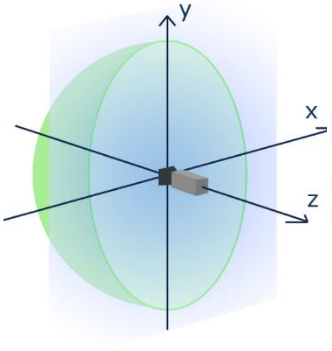
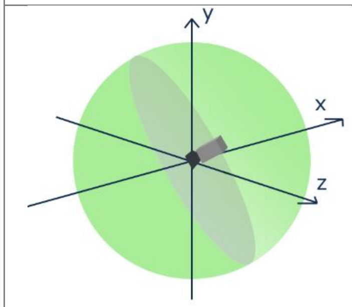
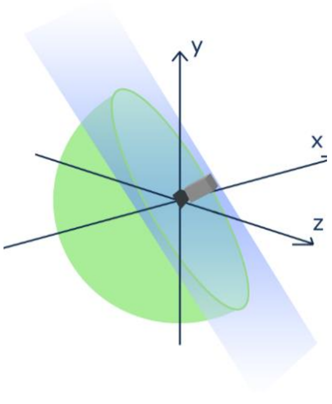
Pan: -180° to +180°  	Perimeter mode - ceiling **360° navigation referential** Origin → facing down **Capture device rotation** pan=0, tilt=-90, roll=0 Perimeter View: Field of view: 2 x 180° Customizable Perimeter View: Field of view: 1 x 360°
Pan: -180° to +180°  	Perimeter mode - ground **360° navigation referential** Origin → facing up **Capture device rotation** pan=0, tilt=90, roll=0 Perimeter View: Field of view: 2 x 180° Customizable Perimeter View: Field of view: 1 x 360°
Pan: -90° to +90° (1) 	Perimeter mode - wall **360° navigation referential** Origin → facing the horizon **Capture device rotation** pan=0, tilt=0, roll=0 Perimeter View: Field of view: 1 x 180° Customizable Perimeter View: Field of view: 1 x 180°

(1) Default values depend on the real Field Of View of the panomorph lens. If the lens Field of View is 190°, the pan will be -95° to +95°.

Navigation Type

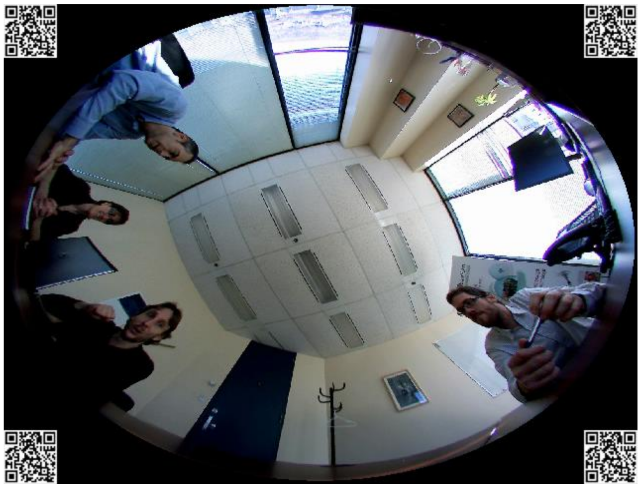
Three types of navigation can be used to change the user experience related to the 360°:

- 360x Panomorph Lens FOV Locked
- 360x360 Stabilized
- 360x360 Stabilized Locked

	The green area represents the area covered by the virtual camera in Wall position (respectively for Ceiling and Ground position). The blue plane shows the limit of the area.	360xFOV Locked – Default mode The virtual camera can only move inside the area defined by the 360° navigation referential selected (wall, ceiling, and ground). The virtual camera is not able to move out of the FOV defined by the Panomorph image even if the capture device moves. If the capture device does not move, we recommend the 360xFOV Locked Navigation Type.
	The green area represents the area covered by the virtual camera. Basically, the library allows the virtual camera to move everywhere.	360x360 Stabilized The virtual camera pans and tilts in the 360° environment without any limits, no matter the movements of the capture device. A black image is displayed outside the image FOV.
	The green area represents the area covered by the virtual camera in Wall position. The blue plane shows the limit of the area. Basically, If the capture device rotates, the area covered by the virtual camera follows the capture device rotation.	360x360 Stabilized Locked The virtual camera moves inside the area defined by the field of view of the panomorph lens. The virtual camera is not able to move out of the FOV limited by the Panomorph image. If the capture device rotation continuously changes (mobile camera for example), using a 360x360 Stabilized Navigation type might be a better choice.

Immervision Enables 2.0 Markers

Immervision Enables 2.0 images contain markers including information.
These markers contain information regarding the parameters of a panomorph lens such as image footprint coordinates, RPL or capture device position, but can also contain others panomorph image parameters such as date, camera localization, capture device rotation etc.



Immervision Enables 2.0 images contain markers including information

Specific information is obtained using its tag name and the function *GetMarkersInfo*.

```
unsigned long GetMarkersInfo(     char /*in*/  *tag,
                                  void /*out*/  *value,
                                  int /*out*/  *nbBytes);
```

Example1:
int nbBytes;
char RPL[6];
unsigned long result = *GetMarkersInfo* (“**RPL**”,(void*)(&**RPL)**, &**nbBytes**);

If the markers contain the tag “RPL”, *value* will contain the RPL string and nbBytes will be equal to 6 as described in the Tag List documentation.

Example2:
int nbBytes;
float value;
unsigned long result = *GetMarkersInfo* (“**Acceleration**”,(void*)(&**value**), &**nbBytes**);

If the markers contain the tag “Acceleration”, *value* will contain a float value and nbBytes will be equal to 1 as described in the Tag List documentation.

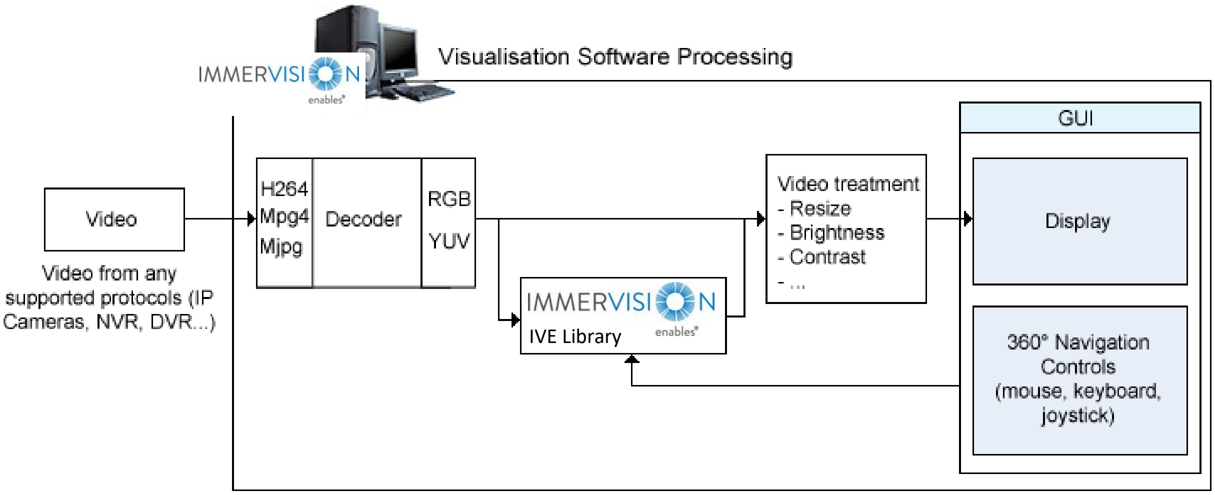
Example3:
int nbBytes;
float value[3];
unsigned long result = *GetMarkersInfo* (“**DeviceOrientation**”,(void*)(&**value**), &**nbBytes**);

If the markers contain the tag “CameraFreeOrientation”, *value* will contain 3 float values and nbBytes will be equal to 3 as described in the Tag List documentation.

USING THE LIBRARY

Using the API (IMV_CameraInterface)

The Immervision Enables library is designed to be integrated on the image processing pipeline through a simple API. Activated just before the final display, the library dewarps the panomorph video frames.


IMMERVISION enables
Visualisation Software Processing
Video
Video from any supported protocols (IP Cameras, NVR, DVR...)
H264
Mpg4
Mjpg
Decoder
RGB
YUV
IMMERVISION
IVE Library
enables
Video treatment
- Resize
- Brightness
- Contrast
- ...
GUI
Display
360° Navigation Controls (mouse, keyboard, joystick)

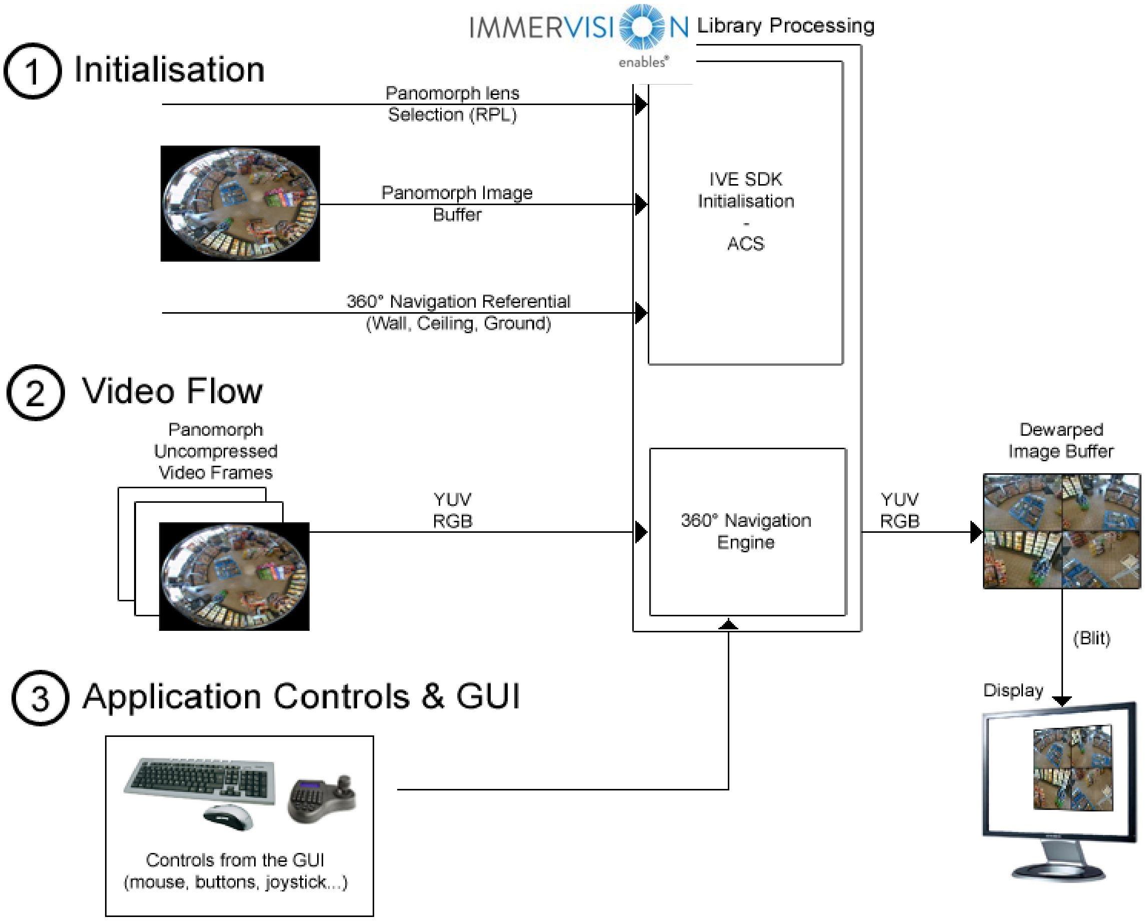
DLL Processing diagram

The Library must take part of the video flow usually between the decoder and the display (blit). The library processes **uncompressed video frames** and **generates uncompressed dewarped picture** (undistorted view).

The library uses the first Panomorph image to calibrate the dewarping engine using the Auto Calibration System (ACS) or use the information in each panomorph image encoded in the Makers

Integration for legacy Immervision Enables devices

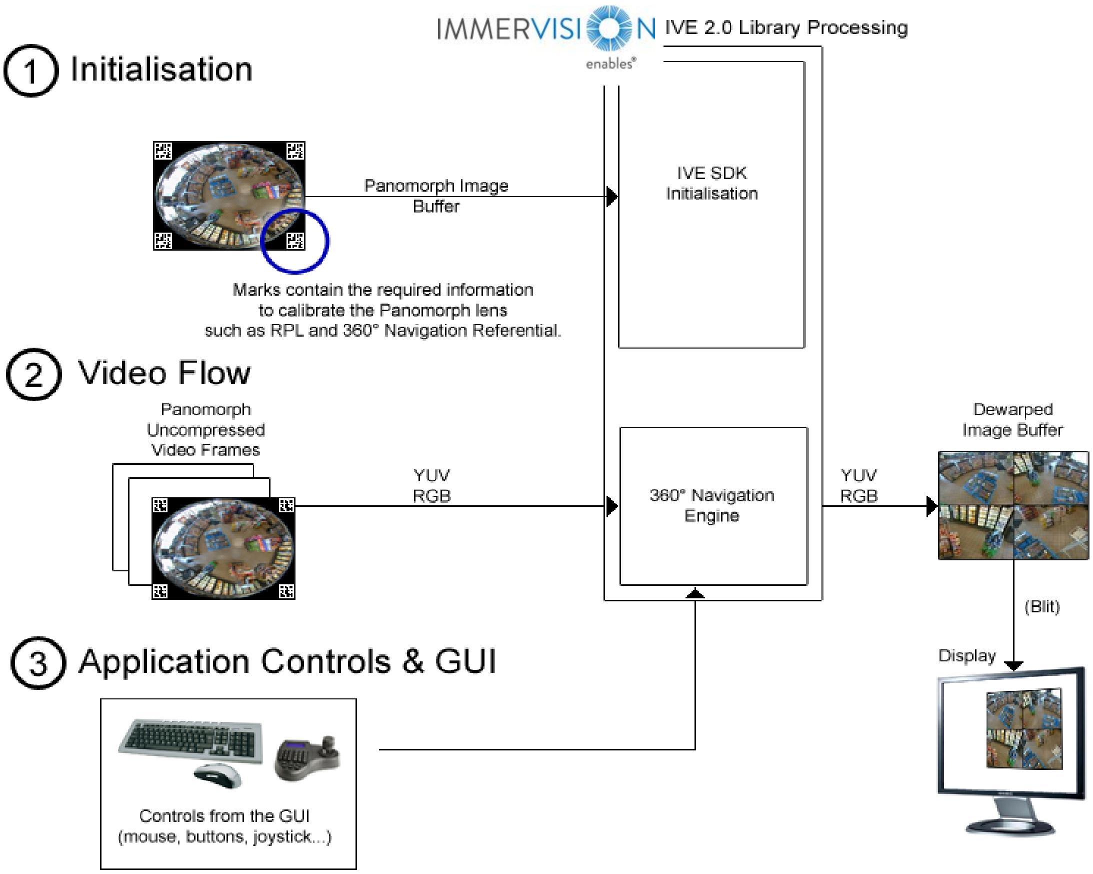
For Immervision Enables legacy devices, the initialization of the library requires a RPL, a Panomorph Image and a 360° Navigation Referential.


IMMERVISION enables® Library Processing
1 Initialisation
Panomorph lens Selection (RPL)
Panomorph Image Buffer
360° Navigation Referential (Wall, Ceiling, Ground)
IVE SDK Initialisation - ACS
2 Video Flow
Panomorph Uncompressed Video Frames
YUV RGB
360° Navigation Engine
YUV RGB
Dewarped Image Buffer
(Blit)
Display
3 Application Controls & GUI
Controls from the GUI (mouse, buttons, joystick...)

Immervision Enables Library Basic Integration

Immervision Enables 2.0 Full Integration

An Immervision Enables 2.0 panomorph image contains the required information to correctly initialize the Immervision Enables 2.0 library without having to indicate the Panomorph Lens RPL of the lens or the 360° Navigation Referential (wall, ceiling, ground) to the library.
With this feature, no more selection of the Panomorph Lens or the 360° Navigation Referential is required from the user to correctly dewarp the Panomorph Image.

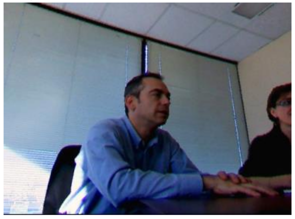
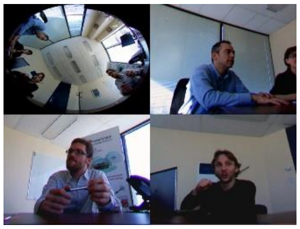
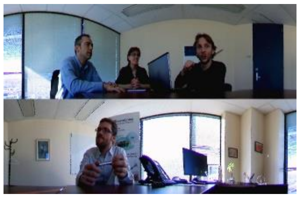
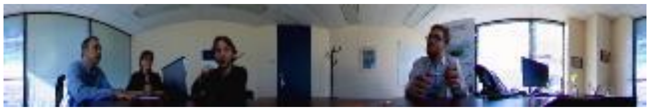

IMMERVISION enables®
IVE 2.0 Library Processing
1 Initialisation
Panomorph Image Buffer
IVE SDK Initialisation
Marks contain the required information to calibrate the Panomorph lens such as RPL and 360° Navigation Referential.
2 Video Flow
Panomorph Uncompressed Video Frames
YUV RGB
360° Navigation Engine
YUV RGB
Dewarped Image Buffer
(Blit)
Display
3 Application Controls & GUI
Controls from the GUI (mouse, buttons, joystick...)

Immervision Enables Library 2.0 Integration

To determine if the panomorph picture contains RPL and 360° Navigation Referential information, use the static function *GetMarkersInfo* described in the Reference-Functions chapter.
If these values exist, you will not need to ask a user to select the RPL or the 360° Navigation Referential.

To know all the information available, please refer to the Immervision Enables 2.0 SDK Tags List Documentation for more information
Note: To have access to the new functionalities offered by the Immervision Enables 2.0 standard, just replace the library v1.x.x.x with the library v2.x.x.x.

Library Input and Output Parameters

Input: Lens RPL	Defines the lens type to dewarp (included in the 2.0 markers) (five character string ex: A0**V)
Input: panomorph video	Uncompressed video (picture buffers) in supported format (see eColorFormat definition) and with a minimal resolution of 640x480.
Input: 360° Navigation Referential	Ceiling, Ground or Wall perspective. (included in the 2.0 markers)
Input: View position	Pan, Tilt and Zoom angle.
Input: View type	Single, Quad and one or two 180° perimeter views.  Single View  Quad View (4 Single views)  Perimeter View  Customizable Perimeter View
Input: picture parameter	Width and height of the output image.
Output: picture buffer	Image calculated by the Library to be displayed by the host application. The output format corresponds to the same input format (see eColorFormat definition).

The API

The API includes two classes and one structure.

One class IMV_Defs declares all the enums used by the library. See REFERENCE-ENUMS for more information.

One class IMV_CameraInterface contains all the library functions. You must create an IMV_CameraInterface object to use the library functionalities and dewarp your videos. See REFERENCE-FUNCTIONS for more information

One structure which describes the input and output video buffers used by the library. It will be discussed in the next paragraph.

The IMV_Buffer structure

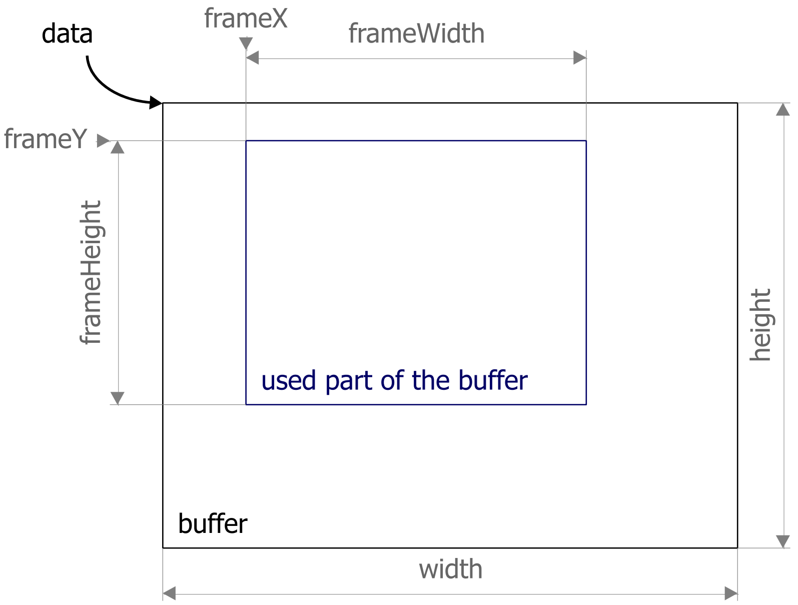
```
typedef struct
{
	unsigned long width;
	unsigned long height;
	unsigned long frameX;
	unsigned long frameY;
	unsigned long frameWidth;
	unsigned long frameHeight;
	unsigned char *data;
} IMV_Buffer ;
```

The library works with two memory buffers: the first one represents the video-in buffer; the second one represents the video-out buffer.
The video-in buffer contains the panomorph video frame data.
The video-out buffer contains the dewarped video frame data.

Buffer structure parameters:

width	Width of the buffer in pixels
height	Height of the buffer in pixels
frameX	X-position of the used part of the buffer in pixels
frameY	Y-position of the used part of the buffer in pixels
frameWidth	Width of the used part of the buffer in pixels
frameHeight	Height of the used part of the buffer in pixels
data	Pointer to the buffer data. Data are stocked in a one dimension array, line by line.

The buffer represents a rectangle surface in the memory that contains the picture to be processed. To facilitate the development, the library allows the buffer to be bigger than the actual used part of the buffer:



The size of buffer data must be: width x height x color_depth. There are some size limitations: each of the values (width, height, frameX, frameY, frameWidth and frameHeight) must be multiples of 4. This allows using standard RGB formats and YUV formats. The supported formats are:

- RGB 16 bits: X1 R5 G5 B5
- RGB 24 bits: R8 G8 B8
- RGB 32 bits: X8 R8 G8 B8
- YV12
- IYUV
- I420
- UYVY
- YUYV
- YUY2
- NV12
- NV21

When a new buffer is passed to the library, some validation checks are made. A buffer is considered invalid if:

- buffer is NULL
- data is NULL
- width or height = 0
- frameWidth or frameHeight = 0
- (frameX + frameWidth) > width
- (frameY + frameHeight) > height
- width, height, frameX, frameY, frameWidth or frameHeight are not multiple of 4.
- the size of the data is not checked. **If size(data) < (width x height x color_depth), the library may crash.**

The buffers are passed to the library through either the SetVideoParams(…), SetInputVideoParams(…) or SetOutputVideoParams(…) functions. If one of the buffer parameters (full-size, size or position) is modified, you must call back one of these functions before processing the video-frame with the Update() function.

The Update() function will only accept a modification of the data content to feed the video-in buffer with the new panomorph video frame for dewarping. It uses this new panomorph video frame to process the data and feed the video-out buffer.

Integration – Step by step

To use the library, you must first instantiate an IMV_CameraInterface object. The library's functions are accessible through its functions.

```
IMV_CameraInterface *camera = new IMV_CameraInterface() ;
```

Initializing the library

In order to dewarp the video stream, you must initialize the library with the SetVideoParams(…) function. As soon as the library is initialized, you can use all the other functions to process the video.

```
//We want to display a 640x480x24bits panomorph video stream on
a 320x240 surface centered in a 400x300 buffer. The camera is
placed in the ceiling and we want to display the dewarped video
in PTZ mode (Single).
//Creation of the buffers
IMV_Buffer *inBuffer, *outBuffer;
inBuffer = new IMV_Buffer;
outBuffer = new IMV_Buffer;

inBuffer->width = 640;
inBuffer->height = 480;
inBuffer->frameX = 0;
inBuffer->frameY = 0;
inBuffer->frameWidth = 640;
inBuffer->frameHeight = 480;
inBbuffer->data = videoFrameData; //videoFrameData is a pointer
on the next video frame

outBuffer->width = 400;
outBuffer->height = 300;
outBuffer->frameX = 40;
outBuffer->frameY = 28;
outBuffer->frameWidth = 320;
outBuffer->frameHeight = 240;
outBuffer->data = displayData ; //displayData is the pointer to
the memory buffer to display

camera->SetLens(“A0**V”); //we indicates the lens type we use

unsigned long iResult = camera->SetVideoParams(
              inBuffer,
              outBuffer,
              IMV_Defs::E_RGB_24 | IMV_Defs::E_OBUF_TOPBOTTOM,
              IMV_Defs::E_VTYPE_PTZ,
              IMV_Defs::E_CPOS_CEILING);
```

```
if (iResult == E_ERR_OK)
{
    //the library is correctly initialized
}
else
{
    //an error occurred
}

camera->SetZoomLimits(40.f,180.f); // we set the zoom limits of
our viewer (the minimum and maximum value)
```

Displaying the video

To process the input video and feed the output buffer with the result, simply call up the Update() function. Each time a new video frame is ready to be displayed in the data of the input stream, just call up the Update() function again to update the output buffer.

```
camera->Update();

//to display it in GDI (assuming that variables are correctly
initialized)
SetDIBitsToDevice( hdc, 0, 0, outBuffer->width,
                outBuffer->height, 0, 0, 0, outBuffer->height,
                outputBuf->data,
                (BITMAPINFO*)(&outHeader),DIB_RGB_COLORS);

...
//Get a new video frame
inBuffer->data = newVideoFrameData ; // newVideoFrameData is a
pointer to the new video frame

camera->Update();  //updates  outBuffer->data  with  the  new
rendering
```

First display

360° navigation

As explained above, depending on the viewing mode, the virtual camera can be moved to display the dewarped viewing area.

- For the Single and Quad modes, pan, tilt and zoom may be applied. For Perimeter mode, pan can be applied when the camera is oriented vertically.
- The samples show how to easily manipulate the virtual camera with a mouse or keyboard.
- This action is performed with the SetPosition(…) function. It allows the user to choose the virtual camera position to the (0,0) absolute position or relative to the latest position.
- To see the results, call up the Update() function to update the output buffer.

The function uses pointers on pan, tilt and zoom. If one of the values exceeds the limits imposed by the virtual camera, the function will modify the variable with the correct value so that you always know the real position of the virtual camera. If you exceed relative positions, the variables are filled with the new absolute position.

```
float pan, tilt, zoom;
pan = 95.f;
tilt = 70.f;
zoom = 90.f;

//set the new position
camera->SetPosition( &pan, &tilt, &zoom);

...

//update the output buffer
camera->Update();
```

Display of the new position

Modification of the buffer parameters

If you need to change one of the buffer parameters (full size, size or position), you should pass the new parameters to the library. For the input buffer, call up the SetInputVideoParams(…) function. For the output buffer, call up the SetOutputVideoParams(…) function. Changes will take effect during the next call up of the Update() function.

```
//in this example, we want the used part of the output buffer to
have the same size as the buffer

IMV_Buffer outBuffer;

outBuffer.width = 400;
outBuffer.height = 300;
outBuffer.frameX = 0;
outBuffer.frameY = 0;
outBuffer.frameWidth = 400;
outBuffer.frameHeight = 300;
outBuffer.data = displayData; //displayData is the pointer to the
memory buffer to display

camera->SetOutputVideoBuffer( &outputBuffer);

...

//update the output buffer
camera->Update() ;
```

New display

Switching between different view modes

The SetViewType(…) function allows you to change the view mode (Single, Quad or Perimeter). Changes will take effect during the next call up of the Update() function.

```
//set the Quad view mode
camera->SetViewType( IMV_Defs::E_VTYPE_QUAD);

...

//update the output buffer
camera->Update();
```

Quad mode is now displayed

360° Navigation Referential

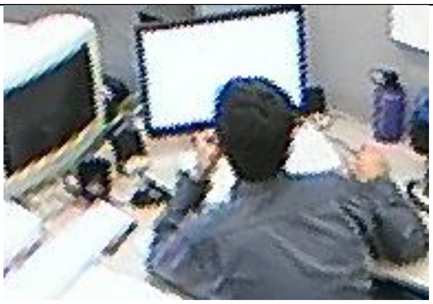
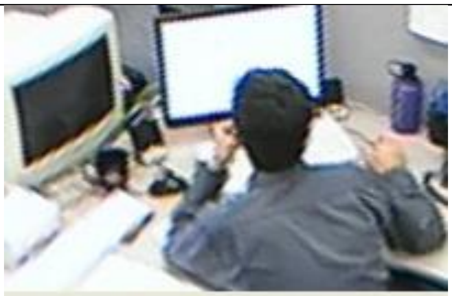
If the capture device with the panomorph lens is placed horizontally and the 360° navigation referential is configured for a capture device placed vertically with the lens up or down, the motion of the virtual camera will not be appropriate for the user. The user must have the option to set the 360° Navigation Referential. This can be done with the SetCameraPosition() function. Changes will take effect during the next call up of the Update() function.

```
//the camera is placed horizontally
camera->SetCameraPosition( IMV_Defs::E_CPOS_WALL);

...

//update the output buffer
camera->Update();
```

Output image filtering (optional)

The library allows you to set the filtering used to enhance the display of your output images rendered by the library.

```
// get the filtering currently used by the library.
unsigned long filtering ;
camera->GetFiltering(&filtering) ;

// set the filtering to the library.
camera->SetFiltering(IMV_Defs::E_FILTER_BILINEAR_ONSTOP);
```

 Picture generated with no filtering	 Picture generated with bilinear filtering

For more details please see page 38.

Closing the library

To close the library, you simply delete the IMV_CameraInterface you created:

```
delete camera;
```

Multithread utilization

For optimisation purpose, the library is not designed to be thread-safe. However, an IMV_CameraInterface object can be used in a multithreaded context. In this case, each IMV_CameraInterface function call must be *protected against concurrent* access from multiple *threads*.

ACS Calibration Parameter

The library allows you to record the ACS Calibration Parameter.
This ACS Calibration Parameter can be used later to initialize the library if it's needed (Calibration during bad lighting conditions (low light) for example).

How and where to call the functions

The first time you run a camera, you must initialize the library with the SetVideoParams(…) function. As soon as the library is initialized, you can use the functions to get and set the Calibration Parameters.

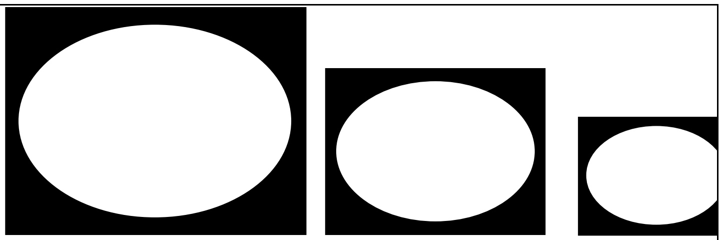
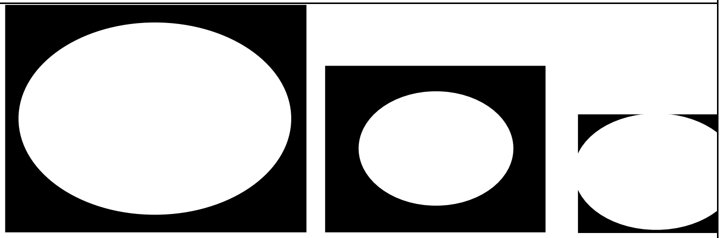
```
// First time we use a system we call SetVideoParams.
unsigned long iResult = camera->SetVideoParams(
                 inBuffer,
                 outBuffer,
                 IMV_Defs::E_RGB_24 | IMV_Defs::E_OBUF_TOPBOTTOM,
                 IMV_Defs::E_VTYPE_PTZ,
                 IMV_Defs::E_CPOS_CEILING);

// if the function has succeeded,
// we get the Calibration Parameters
if (iResult==IMV_Defs::E_ERR_OK)
   char *acsInfo = camera->GetACS();
```

As soon as the Calibration Parameters are recorded, the system can be initialized calling SetACS function before SetVideoParams.

```
// We set the previously recorded Calibration Parameters
camera->SetACS(acsInfo);

// Calling SetVideoParams function does not autoCalibrate the
lens since the information has already been filled out by the
SetACS function.
unsigned long iResult = camera->SetVideoParams( … );
```

Input of the functions

The calibration parameter is independent of the image resolution. For example if the image source is just scaled then you can call GetACS() and SetACS() independently.

Example of image source scaled. The same Calibration Parameter can be used for all of these pictures.	
Example of image source which is not proportionally scaled when the resolution of the capture device is changed. In this case, you have to store the ACS Calibration Parameter values for the different video resolutions.	

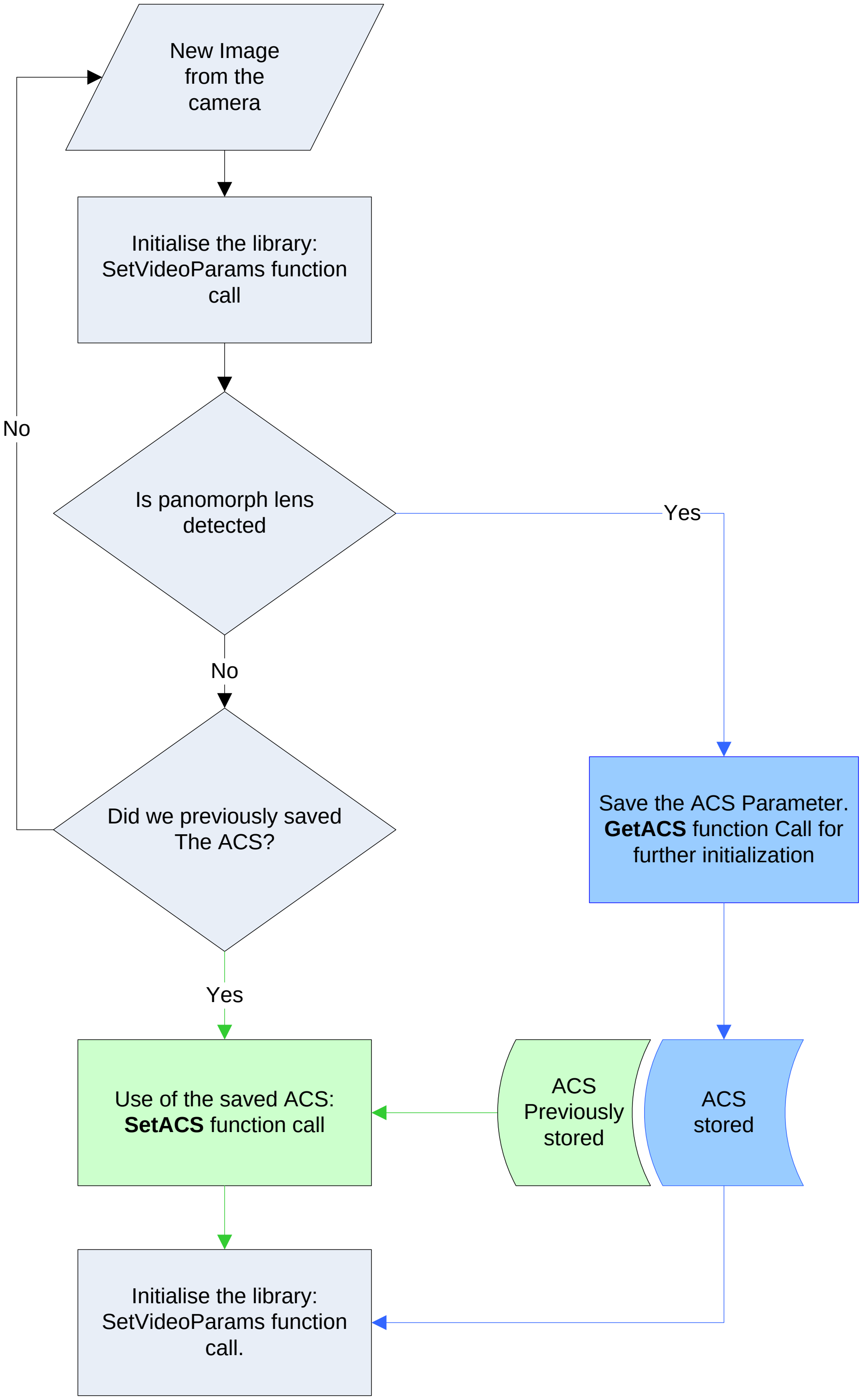
```
// If the resolution of the capture device changes
IMV_Buffer *inBuffer;
inBuffer = new IMV_Buffer;
inBuffer->width = 640;
inBuffer->height = 480;
inBuffer->frameX = 0;
inBuffer->frameY = 0;
inBuffer->frameWidth = 640;
inBuffer->frameHeight = 480;
inBbuffer->data = videoFrameData;

// Calibration Parameters are still used after calling the
SetInputVideoParams function (or SetVideoParams) since the ACS
information has not changed. The resolution is independent by
default.
camera->SetInputVideoParams(inBuffer);
```

How to reset the Calibration Parameter

```
// Set the function with a NULL value to enable back the Auto
Calibration System
camera->SetACS(NULL);

// Calling SetVideoParams function will compute a new
calibration.
unsigned long iResult = camera->SetVideoParams( … );
```

Standard process

If your system requires the use of GetACS and SetACS functions, the recommended process is:


New Image from the camera
Initialise the library: SetVideoParams function call
Is panomorph lens detected
Yes
No
No
Did we previously saved The ACS?
Yes
Save the ACS Parameter. **GetACS** function Call for further initialization
Use of the saved ACS: **SetACS** function call
ACS Previously stored
ACS stored
Initialise the library: SetVideoParams function call.

GetACS function is called when the panomorph lens is detected to save ACS parameters.

Call SetACS function as soon as the panomorph lens is not automatically detected during further initialization.
Note that if the panomorph lens is not automatically detected, this may be not a panomorph lens, the panomorph lens may have been physically shifted or moved or the image quality may have been altered.
Be sure to call SetACS function only when it´s required.

IMV_CameraFlatSurfaceAPIInterface

1- The former IMV_CameraInterface is designed in order to display the flat picture buffer (RGB or YUV) on the software GUI using non-accelerated API (like GDI or Microsoft Windows).
2- With the IMV_CameraFlatSurfaceAPIInterface the library allows to display the same flat picture in 2D using an accelerated Graphic API (OpenGL, DirectX,...).
All the processing and functions calls remains the same, except the flat picture must be obtain each time it's required during 360° navigation.
The export is done in a format compatible with the standard Accelerated Graphic API.

A sample code demonstrates the utilisation of this class in OpenGL (Sample FlatSurface_openGL_Blit).

Benchmarks

Although Immervision Enables library 2.0 includes markers reading and new visual features, the CPU load is quite similar than the previous Immervision Enables library 1.0.

	Movement Processing (pan, Tilt and Zoom)				**Refresh Processing (No movement)**			
	Input picture size VGA (640x480)		Input picture size 1.2Mp (1280x960)		Input picture size VGA (640x480)		Input picture size 1.2Mp (1280x960)	
	Output size 640x480	Output size 320x240	Output size 640x480	Output size 320x240	Output size 640x480	Output size 320x240	Output size 640x480	Output size 320x240
Immervision Enables library 2.0 Core 2 Q6600 2.40Ghz	**4,32ms** (231fps)	**1,95ms** (511fps)	**5,15ms** (194fps)	**2,14ms** (467fps)	**0,35ms** (2830fps)	**0,09ms** (11219fps)	**0,44ms** (2267fps)	**0,12ms** (8378fps)

CPU LOAD on both Immervision Enables library 2.0 and 1.0

CPU Performance based on 100% CPU load. * % CPU load based on fixed 25fps. Note that on a multicore processor, only one core is considered.

REFERENCE – ENUMS

All the enums used by the library are contained in the IMV_Defs class.

ACS Warning codes: eACSStatusCode

These codes are returned by GetACSStatus function to indicate ACS (Auto-Calibration System) status.

Values

E_WAR_ACS_OK	ACS function succeeded.
E_WAR_ACS_NOTDETECTED	The ACS was not able to detect a panomorph image.
E_WAR_ACS_CROPPEDLEFT	The left part of the panomorph image is outside of the frame buffer.
E_WAR_ACS_CROPPEDRIGHT	The right part of the panomorph image is outside of the frame buffer.
E_WAR_ACS_CROPPEDTOP	The top part of the panomorph image is outside of the frame buffer.
E_WAR_ACS_CROPPEDBOTTOM	The bottom part of the panomorph image is outside of the frame buffer.
E_WAR_ACS_NOTCENTERED	The panomorph image is not vertically or horizontally centered
E_WAR_ACS_ROTATED	The panomorph image is too much rotated

Type of view: eBackgroundType

This enum is used to determine which background will be displayed during the navigation.

Values

E_BACK_NONE	By default. No background is displayed. The background remains black.
E_BACK_LINES	White lines are displayed to give a vision of the perspective.

Camera position: eCameraPosition

This enum is used to describe the 360° Navigation Referential of the capture device within its designated space.

Values

E_CPOS_WALL	The 360° Navigation Referential is placed horizontally (wall position)
E_CPOS_CEILING	The 360° Navigation Referential is placed vertically with lens down (ceiling position)
E_CPOS_GROUND	The 360° Navigation Referential is placed vertically with lens up (ground position)

Color format: eColorFormat

This enum is used to describe the color format of the input and output buffers passed on to the library.

Values

E_RGB_16	The color format of the buffers is 16 bits RGB (X1 R5 G5 B5) with Windows style Bitmap order (bottom to top Bitmap)
E_RGB_24	The color format of the buffers is 24 bits RGB (R8 G8 B8) with Windows style Bitmap order (bottom to top Bitmap)
E_RGB_32	The color format of the buffers is 32 bits RGB (X8 R8 G8 B8) with Windows style Bitmap order (bottom to top Bitmap)
E_RGB_16_STD	The color format of the buffers is 16 bits RGB (X1 R5 G5 B5) with standard style Bitmap order (top to bottom Bitmap)
E_RGB_24_STD	The color format of the buffers is 24 bits RGB (R8 G8 B8) with standard style Bitmap order (top to bottom Bitmap)
E_RGB_32_STD	The color format of the buffers is 32 bits RGB (X8 R8 G8 B8) with standard style Bitmap order (top to bottom Bitmap)
E_YUV_YV12	The color format is YUV (YV12 FOURCC code) with Windows style Bitmap order (bottom to top Bitmap)
E_YUV_IYUV	The color format is YUV (IYUV FOURCC code) with Windows style Bitmap order (bottom to top Bitmap)
E_YUV_I420	The color format is YUV (I420 FOURCC code) with Windows style Bitmap order (bottom to top Bitmap)
E_YUV_YV12_STD	The color format is YUV (YV12 FOURCC code) with standard style Bitmap order (top to bottom Bitmap)
E_YUV_IYUV_STD	The color format is YUV (IYUV FOURCC code) with standard style Bitmap order (top to bottom Bitmap)

E_YUV_I420_STD	The color format is YUV (I420 FOURCC code) with standard style Bitmap order (top to bottom Bitmap)
E_YUV_UYVY	The color format is YUV (UYVY FOURCC code) with Windows style Bitmap order (bottom to top Bitmap)
E_YUV_YUYV	The color format is YUV (YUYV FOURCC code) with Windows style Bitmap order (bottom to top Bitmap)
E_YUV_YVYU	The color format is YUV (YVYU FOURCC code) with Windows style Bitmap order (bottom to top Bitmap)
E_YUV_YUY2	The color format is YUV (YUY2 FOURCC code) with Windows style Bitmap order (bottom to top Bitmap)
E_YUV_UYVY_STD	The color format is YUV (UYVY FOURCC code) with standard style Bitmap order (top to bottom Bitmap)
E_YUV_YUYV_STD	The color format is YUV (YUYV FOURCC code) with standard style Bitmap order (top to bottom Bitmap)
E_YUV_YVYU_STD	The color format is YUV (YVYU FOURCC code) with standard style Bitmap order (top to bottom Bitmap)
E_YUV_YUY2_STD	The color format is YUV (YUY2 FOURCC code) with standard style Bitmap order (top to bottom Bitmap)
E_ARGB_32	The color format of the buffers is 32 bits RGB (X8 R8 G8 B8) with Windows style Bitmap order (bottom to top Bitmap) – same as E_RGB_32
E_ARGB_32_STD	The color format of the buffers is 32 bits RGB (X8 R8 G8 B8) with standard style Bitmap order (top to bottom Bitmap)with standard style Bitmap order (top to bottom Bitmap) – same as E_RGB_32_STD
E_RGBA_32	The color format of the buffers is 32 bits RGB (R8 G8 B8 X8) with Windows style Bitmap order (bottom to top Bitmap)
E_RGBA_32_STD	The color format of the buffers is 32 bits RGB (R8 G8 B8 X8) with standard style Bitmap order (top to bottom Bitmap)with standard style Bitmap order (top to bottom Bitmap)
E_YUV_NV12	The color format is YUV (NV12 FOURCC code) with Windows style Bitmap order (bottom to top Bitmap)
E_YUV_NV12_STD	The color format is YUV (NV12 FOURCC code) with Windows style Bitmap order (bottom to top Bitmap)
E_YUV_NV21	The color format is YUV (NV21 FOURCC code) with Windows style Bitmap order (bottom to top Bitmap)
E_YUV_NV21_STD	The color format is YUV (NV21 FOURCC code) with Windows style Bitmap order (bottom to top Bitmap)
E_BGR_16	The color format of the buffers is 16 bits BGR (X1 B5 G5 R5) with Windows style Bitmap order (bottom to top Bitmap)
E_BGR_24	The color format of the buffers is 24 bits BGR (B8 G8 R8) with Windows style Bitmap order (bottom to top Bitmap)
E_BGR_32	The color format of the buffers is 32 bits BGR (X8 B8 G8 R8) with Windows style Bitmap order (bottom to top Bitmap)

E_BGR_16_STD	The color format of the buffers is 16 bits BGR (X1 B5 G5 R5) with standard style Bitmap order (top to bottom Bitmap)
E_BGR_24_STD	The color format of the buffers is 24 bits BGR (B8 G8 R8) with standard style Bitmap order (top to bottom Bitmap)
E_BGR_32_STD	The color format of the buffers is 32 bits BGR (X8 B8 G8 R8) with standard style Bitmap order (top to bottom Bitmap)
E_ABGR_32	The color format of the buffers is 32 bits BGR (X8 B8 G8 R8) with Windows style Bitmap order (bottom to top Bitmap) – same as E_BGR_32
E_ABGR_32_STD	The color format of the buffers is 32 bits BGR (X8 B8 G8 R8) with standard style Bitmap order (top to bottom Bitmap)with standard style Bitmap order (top to bottom Bitmap) – same as E_ BGR _32_STD
E_BGRA_32	The color format of the buffers is 32 bits BGR (B8 G8 R8 X8) with Windows style Bitmap order (bottom to top Bitmap)
E_BGRA_32_STD	The color format of the buffers is 32 bits BGR (B8 G8 R8 X8) with standard style Bitmap order (top to bottom Bitmap)with standard style Bitmap order (top to bottom Bitmap)

Coordinates type: eCoordinates

This enum is used to define if the coordinates (pan, tilt and zoom) as passed to the library are absolute to the center, or relative to the latest position.

Values

E_COOR_ABSOLUTE	Coordinates are absolute
E_COOR_RELATIVE	Coordinates are relative

Error codes: eErrorCode

These codes are returned by each library function to indicate success or errors in the calling up process.

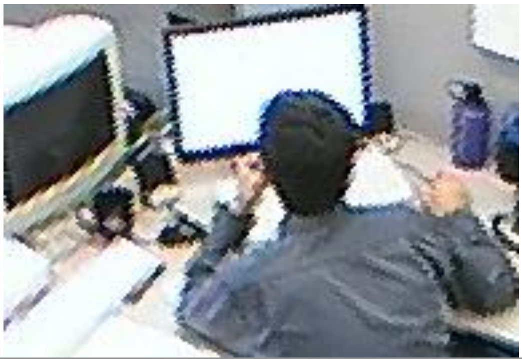
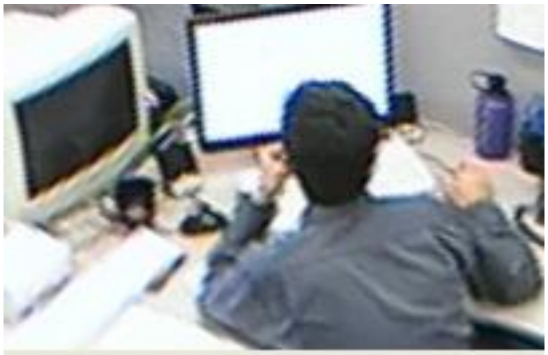
Values

E_ERR_OK	Function succeeded
E_ERR_OUTOFMEMORY	Library is not able to create new objects in memory. This is a very serious error. It is recommended you close the library.
E_ERR_VTYPEINVALID	The view type passed in parameter to the function is invalid. It is not declared in the IMV_Defs::eViewType enum.
E_ERR_CPOSINVALID	The 360° Navigation Referential passed in parameter to the function is invalid. It is not declared in the IMV_Defs::eCameraPosition enum.
E_ERR_COLORINVALID	The color format passed in parameter to the function is invalid. It is not declared in the IMV_Defs::eColorFormat enum.
E_ERR_IBUFINVALID	The input buffer passed in parameter to the function is invalid.
E_ERR_OBUFINVALID	The output buffer passed in parameter to the function is invalid.
E_ERR_INDEXINVALID	The index passed in parameter to the function is invalid. The index is not included in the range allowed.
E_ERR_NOTINITALIZED	This error happens when you call up a library function before initializing the library (call to the SetVideoParam(…) function).
E_ERR_NOTPANOMORPH	A test of the video stream in the input buffer is made to determine if the capture device is using a panomorph lens. If it is not, this error code is returned.
E_ERR_PARAMINVALID	The input parameters are invalid.
E_ERR_NOTALLOWED	The library cannot run this functionality/function.

Filtering Type : eFilterType

This enum is used to define the filtering option used by the renderer

Values

E_FILTER_NONE	This is the default value. • **Low CPU consumption** • High frame rate • No Image enhancement 
E_FILTER_BILINEAR	The filtering of the renderer is set to standard bilinear quality. • **Image enhancement** • Good frame rate 
E_FILTER_BILINEAR_ONSTOP	The filtering of the renderer changes depending on the player movement: - The filtering is set to standard bilinear quality when the player does not move (pan, tilt and zoom do not change) - The filtering is set to Nearest-neighbor quality when the player moves (pan, tilt or zoom changes)

Type of view: eNavigationType

This enum is used to describe how the user can navigate in the panomorph image.

Values

E_NAV_360xFOV_LOCKED	By default. The navigation is limited by the view angle.
E_NAV_360x360_STABILIZED	The navigation is not limited, the user can look everywhere in the 360° environment. Note, a black area appears when the user looks outside the panomorph image field of view.
E_NAV_360x360_STABILIZED_LOCKED	The navigation automatically follows the capture device device movements.

Output Buffer Orientation: eOutputBufferOrientation

This enum is used to tell the library the orientation of the output rendering.

Values

E_OBUF_TOPBOTTOM	The Output buffer has the standard orientation. The first scan line is the top line. (default value)
E_OBUF_BOTTOMUP	The Output buffer is inverted, that means that the first scan line is the bottom line.

Output Buffer Rotation: eOutputBufferRotation

This enum is used to tell the library the rotation of the output rendering.

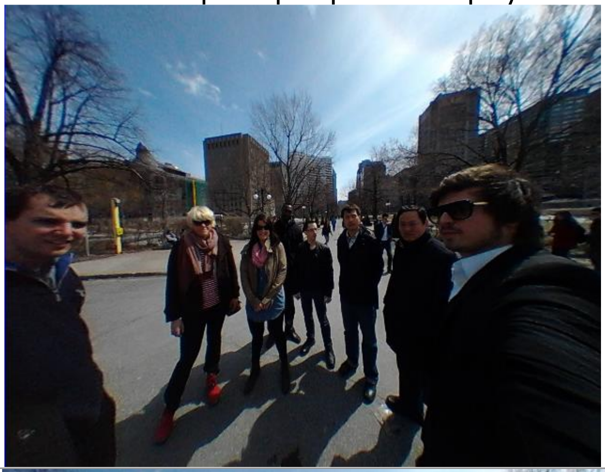
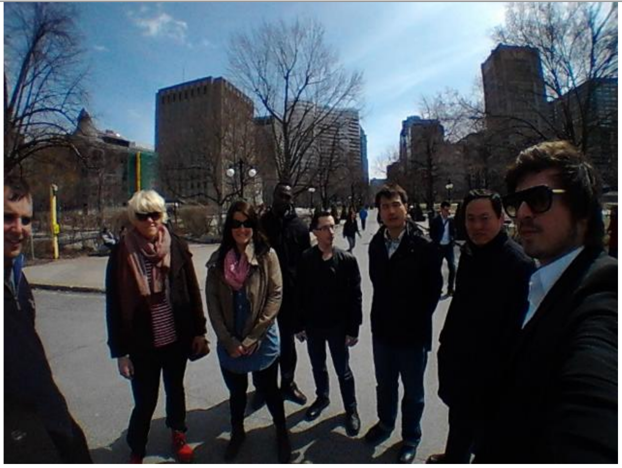
Values

E_OBUF_ROTATION_0	The Output buffer has no rotation. (default value)
E_OBUF_ROTATION_90	The Output buffer has 90° rotation
E_OBUF_ROTATION_180	The Output buffer has 180° rotation
E_OBUF_ROTATION_270	The Output buffer has 270° rotation

Projection Type : eProjectionType

This enum is used to set the projection used by the renderer

Values

E_PROJ_LINEAR	This is the default value. Standard one-point perspective display 
E_PROJ_SCENIC	 The scenic projection enhances the visual proportions of the objects in the scene when the Field of View is widely opened. In this example of a 140° Field of View, the proportion of the people at the edges looks more natural and the people at the center appear closer

Type of view: eViewType

This enum is used to describe the view type that will be used to display the video frames without distortion.

Values

E_VTYPE_PTZ	The Single mode
E_VTYPE_QUAD	The Quad mode
E_VTYPE_PERI	The Perimeter mode
E_VTYPE_PERI_CUSTOM	The fully customisable perimeter mode
E_VTYPE_VERTICAL_SELFIE	The Vertical Selfie mode
E_VTYPE_ZOOM	This parameter associated to a view enables the zoom functionality inside the perimeter views. Example: SetViewType(IMV_Defs::E_VTYPE_PERI | IMV_Defs::E_VTYPE_ZOOM); With this mask, zoom parameter from position (pan, tilt, zoom) will affect the view, and the user will be able to zoom in/out in the output image.

REFERENCE – STRUCTURES

Data buffer: IMV_Buffer

IMV_Buffer is the structure that describes picture data buffers used by the library.

Members

width	Width of the buffer in pixels
height	Height of the buffer in pixels
frameX	X-position of the used part of the buffer in pixels
frameY	Y-position of the used part of the buffer in pixels
frameWidth	Width of the used part of the buffer in pixels
frameHeight	Height of the used part of the buffer in pixels
data	Pointer to the buffer data. Data are stocked in a one dimension array, line by line.

Lens Description: SLensDescription

SLensDescription is the structure that describes a panomorph lens used by the library.

Members

RPL	Pointer to the RPL (Registered Panomorph Lens) of the lens.
Name	Pointer to the commercial name of the lens.

Point: Vertex2D

Vertex2D is the structure that describes a 2D point.

Members

x	X coordinate
y	Y coordinate

REFERENCE – FUNCTIONS

GetACS

```
char *GetACS();
```

Description

This function returns the encoded Calibration Parameters buffer including the ACS information of the system.

Parameters

-

Return codes

If the function succeeds, it returns a pointer to a null terminated string representing encoded calibration parameters;
If the library is not initialized properly (SetVideoParams), returns NULL.

Remarks

- A calibration is unique and is only adapted for one camera module (sensor + lens). Any modification of the system requires a new calibration.
- The returned pointer is valid until the next GetACS-function call or the IMV_CameraInterface object deletion.

GetACSStatus

```
unsigned long GetACSStatus();
```

Description

This function returns the ACS status. Its value is set after the *SetVideoParams* function call. The return value can be a combination of the different ACS Warning Codes listed in the REFERENCE-ENUMS chapter. This combines all warnings in one value (See Remarks below).

Parameters

-

Return codes

All the return codes are written in the REFERENCE-ENUMS chapter.

Remarks

- To get the different warnings, you can use the '&' operation on each ACS Warning Code. Ex:

```
unsigned long result = GetACSStatus();
if (result & IMV_Defs::E_WAR_ACS_ROTATED)
{
  // Warning
}
else
{
// No warning
}
```

GetACSStatusString

```
char *GetAVSStatusString( int eACSStatusCode);
```

Description

This function returns the string corresponding to the eACSStatusCode parameter.
All these strings can be found in the REFERENCE-ENUMS chapter.
For example *GetAVSStatusString (IMV_Defs::E_WAR_ACS_OK)* returns "ACS function succeeded.*"*.
The returned string is a constant and **does not need any deallocation of memory**.

Parameters

eACSStatusCode	Value of the ACS Status code returned by the program.

Return codes

If the eACSStatusCode corresponds on more than 1 warning, only the first warning is returned.

Remarks

-

GetBackground

```
unsigned long GetBackground( unsigned long /*out*/ *backgroundType);
```

Description

This function returns the background type corresponding to the eBackgroundType passed in parameter.
All these values can be found in the REFERENCE-ENUMS chapter.

Parameters

backgroundType	Value of the Background type returned by the program. Values are provided by the IMV_Defs::eBackgroundType enum.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED
The library is not currently initialized.
IMV_Defs::E_ERR_OK
The function succeeded.

Remarks

-

GetCameraPosition

```
unsigned long GetCameraPosition(
                                unsigned long /*out*/ *cameraPosition);
```

Description

This function returns the 360° navigation referential of the camera corresponding to the eCameraPosition passed in parameter: horizontal (against a wall), vertical with lens down (ceiling) or vertical with lens up (ground).

All these values can be found in the REFERENCE-ENUMS chapter.

Parameters

cameraPosition	Value of the 360° navigation referential returned by the program. Values are provided by the IMV_Defs::eCameraPosition enum.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

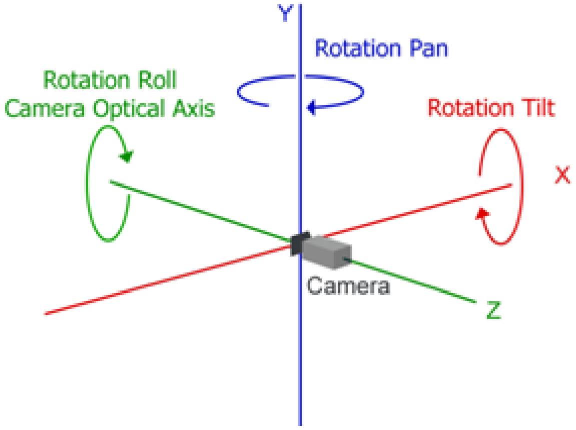
-

GetCameraRotation

```
unsigned long GetCameraRotation( unsigned long /*out*/ *pan,
                                 unsigned long /*out*/ *tilt,
                                 unsigned long /*out*/ *roll);
```

Description

This function returns the camera device rotation.



Parameters

pan	Rotation around Y-axis
tilt	Rotation around X-axis
roll	Rotation around Z-axis (independently from pan and tilt rotation)

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

-

GetDisplayPanLimits

```
unsigned long GetDisplayPanLimits(      float /*out*/ *panMin,
                                        float /*out*/ *panMax);
```

Description

This function returns the displayed pan limits for perimeter modes. Values are expressed in degrees.

The displayed pan corresponds to the actual pan displayed on the final image. This value is calculated by the library when you zoom in/out in a perimeter image.

Parameters

panMin	Minimum displayed pan of the virtual camera.
panMax	Maximum displayed pan of the virtual camera.

Return codes

IMV_Defs::E_ERR_NOTINITIALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

-

GetDisplayTiltLimits

```
unsigned long GetDisplayTiltLimits(     float /*out*/ *tiltMin,
                                        float /*out*/ *tiltMax);
```

Description

This function returns the displayed tilt limits for perimeter modes. Values are expressed in degrees.

The displayed tilt corresponds to the actual tilt displayed on the final image. This value is calculated by the library when you zoom in/out in a perimeter image.

Parameters

tiltMin	Minimum displayed pan of the virtual camera.
tiltMax	Maximum displayed pan of the virtual camera.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

-

GetErrorString

```
char *GetErrorString(int eErrorCode);
```

Description

This function returns the string corresponding to the eErrorCode parameter.
All these strings can be found in the REFERENCE-ENUMS chapter.
For example *GetErrorString(IMV_Defs::E_ERR_OK)* returns *"*Function succeeded*".*
The returned string is a constant and **does not need any deallocation**.

Parameters

eErrorCode	Value of the error code returned by the program.

Return codes

NULL: The index passed in parameter to the function is invalid. The index is not included in the range allowed.

Remarks

-

GetFiltering

```
unsigned long GetFiltering(unsigned long /*out*/ *filtering);
```

Description

This function returns the filtering used by the library renderer.

Parameters

filtering	Filtering used by the library renderer.

Return codes

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

-

GetMarkersInfo - STATIC

```
static unsigned long GetMarkersInfo(
                                  IMV_Buffer /*in*/ *buffer,
                                  unsigned long /*in*/ colorFormat
                                  char /*in*/ *tag,
                                  void /*out*/ *value,
                                  int /*out*/ *nbBytes);
```

Description

This function checks if the buffer passed in parameters contains markers with information about the capture device.

Parameters

buffer	Buffer to check. This buffer and its data must not be NULL and must respect the IMV_Buffer structure.
colorFormat	Color format of the video data contained in the buffer. Accepted values are provided by the IMV_Defs::eColorFormat enum.
tag	String information to get. “RPL” or “CameraFixedOrientation” for example.
value	Value corresponding to the tag passed in parameter found in the markers.
nbBytes	Number of bytes used to store the value.

Return codes

IMV_Defs::E_ERR_PARAMINVALID
The tag in not found in the markers

IMV_Defs::E_ERR_COLORINVALID
The value of colorFormat is not declared in the eColorFormat enum.

IMV_Defs::E_ERR_IBUFINVALID
The input buffer is invalid.

IMV_Defs::E_ERR_OK
Function succeeded.

Remarks

- This function is static and can be called before initializing the library with SetVideoParams to get some information about the system.
  For example, this can get the lens RPL in order to automatically initialize the library without asking to the user.

Example1:

```
int nbBytes;
char RPL[6];
unsigned long result = GetMarkersInfo (“RPL”,(void*)(&RPL), &nbBytes);
```

If the markers contain the tag “RPL”, *value* will contain the RPL string and nbBytes will be equal to 6 as described in the Tag List documentation.

Example2:

```
int nbBytes;
float value;
unsigned long result = GetMarkersInfo (“Acceleration”,(void*)(&value), &nbBytes);
```

If the markers contain the tag “Acceleration”, *value* will contain a float value and nbBytes will be equal to 1 as described in the Tag List documentation.

Example3:

```
int nbBytes;
float value[3];
unsigned long result = GetMarkersInfo (“DeviceOrientation”,(void*)(&value),
&nbBytes);
```

If the markers contain the tag “CameraFreeOrientation”, *value* will contain 3 float values and nbBytes will be equal to 3 as described in the Tag List documentation.

GetMarkersInfo

```
unsigned long GetMarkersInfo(     char /*in*/  *tag,
                                  void /*out*/  *value,
                                  int /*out*/  *nbBytes);
```

Description

This function checks if the buffer passed in parameters contains markers with information about the capture device.

Parameters

tag	String information to get. "RPL" or "CameraFixedOrientation" for example.
value	Value corresponding to the tag passed in parameter found in the markers.
nbBytes	Number of bytes used to store the value.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_PARAMINVALID

The tag in not found in the markers

IMV_Defs::E_ERR_OK

Function succeeded.

Remarks

- The library must be initialized.

GetNavigationType

```
unsigned long GetNavigationType(
                                  unsigned long /*out*/ *navigationType);
```

Description

This function returns the navigation type corresponding to the eNavigationType passed in parameter.

All these values can be found in the REFERENCE-ENUMS chapter.

Parameters

navigationType	Value of the navigation type returned by the program. Values are provided by the IMV_Defs::eNavigationType enum.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

-

GetPanLimits

```
unsigned long GetPanLimits( float /*out*/ *panMin,
                            float /*out*/ *panMax);
```

Description

This function returns the pan limits for perimeter modes. Values are expressed in degrees.

Parameters

panMin	Minimum pan of the virtual camera.
panMax	Maximum pan of the virtual camera.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

-

GetPosition

```
unsigned long GetPosition( float /*out*/ *pan,
                            float /*out*/ *tilt,
                            float /*out*/ *zoom,
                            unsigned long viewIndex = 1);
```

Description

This function returns the position (pan, tilt and zoom) of the virtual camera. If the virtual camera is a Quad, use viewIndex to choose the index of the virtual camera you want to get back into a defined position.

Parameters

pan	Will be filled with the current pan of the virtual camera
tilt	Will be filled with the current tilt of the virtual camera
zoom	Will be filled with the current zoom of the virtual camera
viewIndex	Index of the virtual camera for the Quad mode. See the SetPosition(…) function for more information.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_INDEXINVALID

The virtual camera index is out of range. (1 to 4)

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

-

GetPosition

```
unsigned long GetPosition( float /*out*/ *pan,
                            float /*out*/ *tilt,
                            float /*out*/ *roll,
                            float /*out*/ *zoom,
                            unsigned long viewIndex = 1);
```

Description

This function returns the position (pan, tilt, rol and zoom) of the virtual camera. If the virtual camera is a Quad, use viewIndex to choose the index of the virtual camera you want to get back into a defined position.

Parameters

pan	Will be filled with the current pan of the virtual camera
tilt	Will be filled with the current tilt of the virtual camera
roll	Will be filled with the current roll of the virtual camera
zoom	Will be filled with the current zoom of the virtual camera
viewIndex	Index of the virtual camera for the Quad mode. See the SetPosition(…) function for more information.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_INDEXINVALID

The virtual camera index is out of range. (1 to 4)

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

-

GetProjectionType

```
unsigned long GetProjectionType(
                            unsigned long /*out*/ *projectionType,
                            float /*out*/ *strength);
```

Description

This function returns the projection type corresponding to the eProjectionType passed in parameter.

All these values can be found in the REFERENCE-ENUMS chapter.

Parameters

projectionType	Value of the projection type returned by the program. Values are provided by the IMV_Defs::eProjectionType enum.
strength	Strength of the projection. The value is comprised between 0 and 1.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

-

GetTiltLimits

```
unsigned long GetTiltLimits(      float /*out*/ *tiltMin,
                                  float /*out*/ *tiltMax);
```

Description

This function returns the tilt limits for perimeter modes. Values are expressed in degrees.

Parameters

tiltMin	Minimum tilt of the virtual camera.
tiltMax	Maximum tilt of the virtual camera.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

-

GetVersion

```
static const char* GetVersion();
```

Description

This function returns the library version.
This is a constant and doesn’t need any deallocation.
This function is static and can be used without initializing the camera object.

Parameters

-

Return codes

-

Remarks

-

GetViewType

```
unsigned long GetViewType(unsigned long /*out*/ *viewType);
```

Description

This function returns the view type corresponding to the eViewType passed in parameter. : Single mode, Quad mode or Perimeter modes.
All these values can be found in the REFERENCE-ENUMS chapter.

Parameters

viewtype	Value of the view type returned by the program. Values are provided by the IMV_Defs::eViewType enum.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED
The library is not currently initialized.
IMV_Defs::E_ERR_OK
The function succeeded.

Remarks

-

GetZoomLimits

```
unsigned long GetZoomLimits(      float /*out*/ *zoomMin,
                                  float /*out*/ *zoomMax);
```

Description

This function returns the zoom limits for Single and Quad modes. Values are expressed in degrees.

Parameters

zoomMin	Minimum zoom of the virtual camera.
zoomMax	Maximum zoom of the virtual camera.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

-

RestoreMarkersReading

```
unsigned long ResetMarkersReading();
```

Description

Information included in markers is used by default.

Note: Some parameters can be overridden calling the appropriate function. This function restores the Marker parameter reading.

Parameters

-

Return codes

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

The two parameters that can be overridden and reset are:

- CameraPosition
- CameraRotation

SetACS

```
unsigned long SetACS(char /*in*/ *acs);
```

Description

This function calibrates the library with a previous calibration parameter from GetACS.

Parameters

acs	Calibration Parameters of the lens to be used.

Return codes

IMV_Defs::E_ERR_PARAMINVALID

The calibration parameter is invalid. This parameter must come from GetACS.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

After using SetACS, the library stops performing calibration.
To re-enable automatic calibration, use SetACS with a NULL Parameter.

SetBackground

```
unsigned long SetBackground(unsigned long backgroundType);
```

Description

This function allows the user to display a background when navigating outside the field of view of the panomorph.

Parameters

backgroundType	Background type choice. Accepted values are provided by the IMV_Defs::eBackgroundType enum.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED
The library is not currently initialized.

IMV_Defs::E_ERR_PARAMINVALID
The parameter is invalid.

IMV_Defs::E_ERR_OK
The function succeeded.

Remarks

- When the navigationType is set to IMV_Defs::E_NAV_FIXED, the background is never displayed.
- The background disappears when the field of view of the virtual camera is higher than ≈140 °.

SetCameraPosition

```
unsigned long SetCameraPosition(unsigned long cameraPosition);
```

Description

This function allows the user to configure the 360° Navigation Referential of the capture device: horizontal (against a wall), vertical with lens down (ceiling) or vertical with lens up (ground). The changes will take effect during the next call up of the Update() function.

Parameters

cameraPosition	Camera position corresponds to the 360° Navigation Referential. Accepted values are provided by the IMV_Defs::eCameraPosition enum.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_CPOSINVALID

The value of cameraPosition is not declared in the eCameraPosition enum.

IMV_Defs::E_ERR_OUTOFMEMORY

Fatal Error: the computer does not have enough memory to continue running the library and/or other programs. Close the library immediately.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

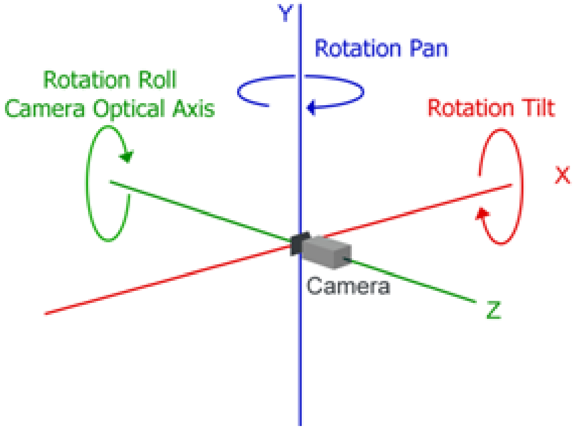
-

SetCameraRotation

```
unsigned long SetCameraRotation( float *pan,
                                 float *tilt,
                                 float *roll);
```

Description

This function allows the user to set the physical rotation of the capture device.



Parameters

pan	Rotation around Y-axis
tilt	Rotation around X-axis
roll	Rotation around Z-axis (independently from pan and tilt rotation)

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

-

SetFiltering

```
unsigned long SetFiltering(unsigned long /*in*/ filtering);
```

Description

This function sets the filtering of the library renderer.

Parameters

filtering	Filtering to be used by the library renderer.

Return codes

IMV_Defs::E_ERR_PARAMINVALID

The value of filtering is not declared in the eFilterType enum

IMV_Defs::E_ERR_NOTALLOWED

The filtering in parameter is not available with the color format used by the library. In this case, the default filtering value is set (IMV_Defs::E_FILTER_NONE)

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

-

SetInputVideoParams

```
unsigned long SetInputVideoParams(IMV_Buffer /*in*/ *inputBuffer);
```

Description

This function replaces the current input buffer by the one passed in parameter. The change will be applied on the next call up of the Update() function.

Parameters

inputBuffer	New input buffer used by the library. This buffer and its data must not be NULL and must respect the IMV_Buffer structure.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_IBUFINVALID

The new input buffer is invalid.

IMV_Defs::E_ERR_NOTPANOMORPH

The input buffer does not contain a video frame from a capture device with an Immervision panomorph lens. The library cannot dewarp it.

IMV_Defs::E_ERR_OUTOFMEMORY

Fatal Error: the computer does not have enough memory to continue running the library and/or other programs. Close the library immediately.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

-

SetLens

```
unsigned long SetLens(char /*in*/ *RPL);
```

Description

This function initializes the library with the selected lens profile.

Parameters

RPL	RPL of the lens to be used in the library.

Return codes

IMV_Defs::E_ERR_OBUFINVALID

The RPL is invalid.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

- See also *StaticGetLensDescriptionCount* and *StaticGetLensDescription* functions.

SetNavigationType

```
unsigned long SetNavigationType(
                                unsigned long /*in*/ navigationType);
```

Description

This function allows the user to change the navigation mode.

Parameters

navigationType	New navigation mode. Accepted values are provided by the IMV_Defs::eNavigationType enum.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_PARAMINVALID

The value of viewType is not declared in the eNavigationType enum.

IMV_Defs::E_ERR_OUTOFMEMORY

Fatal Error: the computer does not have enough memory to continue running the library and/or other programs. Close the library immediately.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

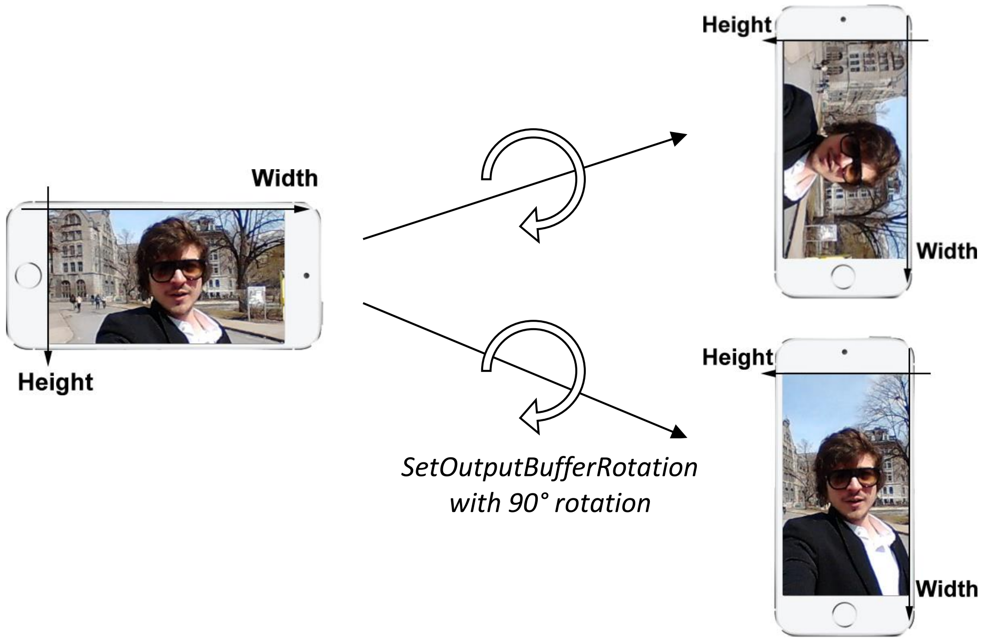
-

SetOutputBufferRotation

```
unsigned long SetOutputBufferRotation(
                      unsigned long /*in*/ outputBufferRotation);
```

Description

This function rotates the outputBuffer. This function must be used only in particular case when the application does not rotate properly the outputBuffer.



Parameters

outputBufferRotation	New rotation value. Accepted values are provided by the IMV_Defs::eOutputBufferRotation enum.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED
The library is not currently initialized.

IMV_Defs::E_ERR_INDEXINVALID
The outputBufferRotation is invalid.

IMV_Defs::E_ERR_OK
The function succeeded.

Remarks

-

SetOutputVideoParams

```
unsigned long SetOutputVideoParams(
                                    IMV_Buffer /*in/out*/ *outputBuffer);
```

Description

This function replaces the current output buffer by the one passed in parameter to the function. The change will take effect during the next call up of the Update() function.

Parameters

outputBuffer	New output buffer used by the library. This buffer must not be NULL.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_OBUFINVALID

The new output buffer is invalid.

IMV_Defs::E_ERR_OUTOFMEMORY

Fatal Error: the computer does not have enough memory to continue running the library and/or other programs. Close the library immediately.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

-

SetPanLimits

```
unsigned long SetPanLimits( float panMin, float panMax);
```

Description

This function allows the user to change the pan limits in Perimeter modes. Values are expressed in degrees.

Parameters

panMin	Minimum pan of the virtual camera. By default Minimum pan is automatically defined by the RPL of the lens used.
panMax	Maximum pan of the virtual camera. By default Maximum pan is automatically defined by the RPL of the lens used.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

SetPosition

```
unsigned long SetPosition( float /*in/out*/ *pan,
                           float /*in/out*/ *tilt,
                           float /*in/out*/ *zoom,
                           unsigned long position =
                                         IMV_Defs::E_COOR_ABSOLUTE,
                           unsigned long viewIndex=1);
```

Description

This function sets the position of the virtual camera:

- for Single mode: sets the pan, tilt and zoom.
- for Quad mode: sets the pan, tilt and zoom for the viewIndex virtual camera.
- for Perimeter mode: sets the pan only when the capture device is positioned vertically.

Values are all expressed in degrees.

Changes will take effect during the next call up of the Update() function.

Parameters

pan	New pan value. If the function succeeds, it returns the absolute value of the pan reached.
tilt	New tilt value. If the function succeeds, it returns the absolute value of the tilt reached.
zoom	New zoom value. If the function succeeds, it returns the absolute value of the zoom reached.
position	Informs the library if the coordinates given for pan, tilt and zoom are absolute to the center of the view or relative to the latest position. Accepted values are provided by the IMV_Defs::eCoordinates enum. By default, the value is IMV_Defs::E_COOR_ABSOLUTE. For more information about the virtual camera coordinates, please refer to the chapters "Using the library/Using the API/The different view modes".
viewIndex	The index of the virtual camera to update. This value is only used by the Quad mode to determine which camera to update. This value must be between 1 and 4. See remark section to know which value corresponds to which virtual camera.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_INDEXINVALID

The virtual camera index is out of range. (1 to 4)

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

The Quad mode displays 4 Single virtual cameras simultaneously. The indexes of the virtual cameras are as follows:

1	**2**
3	**4**

SetPosition

```
unsigned long SetPosition(  float /*in/out*/ *pan,
                            float /*in/out*/ *tilt,
                            float /*in/out*/ *roll,
                            float /*in/out*/ *zoom,
                            unsigned long position =
                                        IMV_Defs::E_COOR_ABSOLUTE,
                            unsigned long viewIndex=1);
```

Description

This function sets the position of the virtual camera.
This function is specifically useful for phone or head gesture movement.
The *roll* parameter allows the user to keep the scene straight, while rotating the device in the optical axis device.

Values are all expressed in degrees.

Changes will take effect during the next call up of the Update() function.

Parameters

pan	New pan value. If the function succeeds, it returns the absolute value of the pan reached.
tilt	New tilt value. If the function succeeds, it returns the absolute value of the tilt reached.
Roll	New roll value. If the function succeeds, it returns the absolute value of the roll reached.
zoom	New zoom value. If the function succeeds, it returns the absolute value of the zoom reached.
position	Informs the library if the coordinates given for pan, tilt and zoom are absolute to the center of the view or relative to the latest position. Accepted values are provided by the IMV_Defs::eCoordinates enum. By default, the value is IMV_Defs::E_COOR_ABSOLUTE. For more information about the virtual camera coordinates, please refer to the chapters "Using the library/Using the API/The different view modes".
viewIndex	The index of the virtual camera to update. This value is only used by the Quad mode to determine which camera to update. This value must be between 1 and 4. See remark section to know which value corresponds to which virtual camera.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_INDEXINVALID

The virtual camera index is out of range. (1 to 4)

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

The Quad mode displays 4 Single virtual cameras simultaneously. The indexes of the virtual cameras are as follows:

1	**2**
3	**4**

SetProjectionType

```
unsigned long SetProjectionType( unsigned long projectionType,
                                 float /*in/out*/ *strength);
```

Description

This function allows the user to change the type of projection.

Parameters

projectionType	New navigation mode. Accepted values are provided by the IMV_Defs::eProjectionType enum.
strength	Strength of the projection. The value is comprised between 0 and 1.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_NAVTYPEINVALID

The value of viewType is not declared in the eProjectionType enum.

IMV_Defs::E_ERR_OUTOFMEMORY

Fatal Error: the computer does not have enough memory to continue running the library and/or other programs. Close the library immediately.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

-

SetThreadCount

```
unsigned long SetThreadCount(int nbThreads);
```

Description

This function enables multithreading for the rendering.

Parameters

nbThreads	Number of threads to be created. The value is comprised between 0 and 32. If nbThreads <2, multithreading is disabled.

Return codes

IMV_Defs::E_ERR_PARAMINVALID

The thread count is invalid

IMV_Defs::E_ERR_NOTALLOWED

This function can't be used with IMV_CameraFlatSurfaceInterface class since this class doesn't do software rendering

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

SetTiltLimits

```
unsigned long SetTiltLimits(float tiltMin, float tiltMax);
```

Description

This function allows the user to change the pan limits for Perimeters mode. Values are expressed in degrees.

Parameters

tiltMin	Minimum tilt of the virtual camera. By default Minimum tilt is automatically defined by the RPL of the lens used.
tiltMax	Maximum tilt of the virtual camera. By default Maximum tilt is automatically defined by the RPL of the lens used.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

SetVideoParams

```
unsigned long SetVideoParams(    IMV_Buffer /*in/out*/ *inputBuffer,
                                 IMV_Buffer /*in/out*/ *outputBuffer,
                                 unsigned long colorFormat,
                                 unsigned long viewType,
                                 unsigned long cameraPosition);
```

Description

This function initializes the library and all its parameters. It must be called up at least once before all other non-static functions.

Parameters

inputBuffer	Input buffer used by the library. This buffer and its data must not be NULL and must respect the IMV_Buffer structure.
outputBuffer	Output buffer used by the library. This buffer must not be NULL.
colorFormat	Color format of the video data contained in the buffers. Accepted values are provided by the IMV_Defs::eColorFormat enum.
viewType	View mode to display: Single, Quad or Perimeter. Accepted values are provided by the IMV_Defs::eViewType and IMV_Defs:: eOutputBufferOrientation enum.
cameraPosition	Camera position corresponds to the 360° Navigation Referential. Accepted values are provided by the IMV_Defs::eCameraPosition enum. If the panomorph image contains Markers, this parameter is not taken into account. This value is overridden by the value included in the markers. Only the value in the markers is considered. Call the following function to get the value included in the markers: *GetMarkersInfo*(“**cameraFixedOrientation**”,(void*)(&**value**),&**nbBytes**);

Return codes

IMV_Defs:: E_ERR_IBUFINVALID

The input buffer is invalid.

IMV_Defs::E_ERR_NOTPANOMORPH

The input buffer does not contain a video frame from a capture device with an Immervision panomorph lens. The library cannot dewarp it.

IMV_Defs::E_ERR_OBUFINVALID

The output buffer is invalid.

IMV_Defs::E_ERR_COLORINVALID

The value of colorFormat is not declared in the eColorFormat enum.

IMV_Defs::E_ERR_VTYPEINVALID

The value of viewType is not declared in the eViewType enum.

IMV_Defs::E_ERR_CPOSINVALID

The value of cameraPosition is not declared in the eCameraPostion enum.

IMV_Defs::E_ERR_OUTOFMEMORY

Fatal Error: the computer does not have enough memory to continue running the library and/or other programs. Close the library immediately.

IMV_Defs::E_ERR_OK

The function succeeded.

SetViewType

```
unsigned long SetViewType(unsigned long viewType);
```

Description

This function allows the user to change the view mode: Single mode, Quad mode or Perimeter mode.

Parameters

viewType	New view mode. Accepted values are provided by the IMV_Defs::eViewType enum.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_VTYPEINVALID

The value of viewType is not declared in the eViewType enum.

IMV_Defs::E_ERR_OUTOFMEMORY

Fatal Error: the computer does not have enough memory to continue running the library and/or other programs. Close the library immediately.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

- If you set viewType=IMV_Defs::E_VTYPE_QUAD position your four (4) views before calling the Update() function.

SetZoomLimits

```
unsigned long SetZoomLimits(float zoomMin, float zoomMax);
```

Description

This function allows the user to change the zoom limits for Single and Quad modes. Values are expressed in degrees.

Parameters

zoomMin	Minimum zoom of the virtual camera must be greater than 5°.
zoomMax	Maximum zoom of the virtual camera must be less than 140° or only set at 180°.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

- The *zoomMax* set at 180° allows the Single or Quad modes viewer to zoom out to the full 360° Panomorph view.

StaticGetLensDescription

```
static const SLensDescription* StaticGetLensDescription();
```

Description

This function allows enumerating the different RPL numbers supported by the library.
It returns an array of SLensDescription structures, each structure containing a different RPL.
The size of the array is obtained by calling the *StaticGetLensDescriptionCount* function.

Example:

```
const SLensDescription* lenses = IMV_CameraInterface::StaticGetLensDescription();

for (int a=0;a<IMV_CameraInterface::StaticGetLensDescriptionCount();a++)
{
    // Add RPL to our RPL selection list.
    SendDlgItemMessage(hDlg, IDC_LIST1, LB_ADDSTRING, 0, (LPARAM) lenses[a].RPL);
}
Delete []lenses;
```

Parameters

-

Return codes

-

Remarks

- The Name represents the commercial name of the lens.
- See also *StaticAddLens* and *StaticGetLensDescriptionCount*.
- This results to a generated array and you have to delete this array from your side.

StaticGetLensDescriptionCount

```
static int StaticGetLensDescriptionCount();
```

Description

This function returns the number of lens supported by the library.

Parameters

-

Return codes

-

Remarks

- See also *StaticAddLens* and *StaticGetLensDescriptionCount* functions.

Update

```
unsigned long Update();
```

Description

This function updates the output buffer content according to the new parameters:

- data in the input buffer
- position: pan, tilt and zoom
- 360° navigation referential
- camera type

Parameters

-

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

- Video-in and video-out buffers must be initialized (!= Null) and allocated (size = width*height*Color_depth). Their data must respect the IMV_Buffer structure requirement.

This function manipulates the input and output buffer memory. The input and output parameter must be correctly allocated by the application.

GetPositionFromInputVideoPoint

```
unsigned long GetPositionFromInputVideoPoint(
    int xInputVideo,
    int yInputVideo,
    float /*out*/ *pan,
    float /*out*/ *tilt);
```

Description

This function computes the position (pan, tilt) of a selected point in the input buffer. The position could be used in the SetPosition function to center the virtual camera on the selected point.

Parameters

xInputVideo	X coordinate of the targeted point in the input buffer
yInputVideo	y coordinate of the targeted point in the input buffer
pan	If the function succeeds, it returns the absolute value of the targeted point pan.
tilt	If the function succeeds, it returns the absolute value of the targeted point tilt.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED
The library is not currently initialized.

IMV_Defs::E_ERR_PARAMINVALID
The input parameters are invalid.

IMV_Defs::E_ERR_OK
The function succeeded.

Remarks

- Video-in buffer must be initialized (!= Null) and allocated (size = width*height*Color_depth). Its data must respect the IMV_Buffer structure requirement.

To avoid visualisation misunderstanding, the Pan and tilt coordinates must be used in SetPosition() function with a zoom factor smaller than 140°.

GetPositionFromOutputVideoPoint

```
unsigned long GetPositionFromOutputVideoPoint(
    int xInputVideo,
    int yInputVideo,
    float /*out*/ *pan,
    float /*out*/ *tilt);
```

Description

This function computes the position (pan, tilt) of a selected point in the output buffer. The position could be used in the SetPosition function to center the virtual camera on the selected point.

Parameters

xInputVideo	X coordinate of the targeted point in the input buffer
yInputVideo	y coordinate of the targeted point in the input buffer
pan	If the function succeeds, it returns the absolute value of the targeted point pan.
tilt	If the function succeeds, it returns the absolute value of the targeted point tilt.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_PARAMINVALID

The input parameters are invalid.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

- Video-in and video-out buffers must be initialized (!= Null) and allocated (size = width*height*Color_depth). Their data must respect the IMV_Buffer structure requirement.

To avoid visualisation misunderstanding, the Pan and tilt coordinates must be used in SetPosition() function with a zoom factor smaller than 140°.

GetPositionFromInputVideoPolygon

```
unsigned long GetPositionFromInputVideoPolygon(
    int nbPts,
    int /*in*/ *xInputVideo,
    int /*in*/ *yInputVideo,
    float /*out*/ *pan,
    float /*out*/ *tilt,
    float /*out*/ *zoom);
```

Description

This function computes the position (pan, tilt, zoom) of a selected polygon in the input buffer. The position could be used in the SetPosition function to center the current virtual camera on the area delimited by the polygon. The zoom is computed using the target polygon size to zoom the virtual camera only on the area delimited by the polygon.

Parameters

nbPts	Size of xInputVideo and yInputVideo
xInputVideo	Array of X coordinate of each polygon vertex
yInputVideo	Array of X coordinate of each polygon vertex
pan	If the function succeeds, it returns the absolute value of the targeted polygon pan.
tilt	If the function succeeds, it returns the absolute value of the targeted polygon tilt.
zoom	If the function succeeds, it returns the absolute value of the targeted polygon zoom.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_PARAMINVALID

The input parameters are invalid

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

- Video-in buffer must be initialized (!= Null) and allocated (size = width*height*Color_depth). Its data must respect the IMV_Buffer structure requirement.

To avoid visualisation misunderstanding, the zoom coordinates max value is smaller than 140°.

GetPositionFromOutputVideoPolygon

```
unsigned long GetPositionFromOutputVideoPolygon(
    int nbPts,
    int /*in*/ *xOutputVideo,
    int /*in*/ *yOutputVideo,
    float /*out*/ *pan,
    float /*out*/ *tilt,
    float /*out*/ *zoom);
```

Description

This function computes the position (pan, tilt, zoom) of a selected polygon in the output buffer. The position could be used in the SetPosition function to center the current virtual camera on the area delimited by the polygon. The zoom is computed using the target polygon size to zoom the virtual camera only on the area delimited by the polygon.

Parameters

nbPts	Size of xInputVideo and yInputVideo
xOutputVideo	Array of X coordinates of each polygon vertex
yOutputVideo	Array of Y coordinates of each polygon vertex
pan	If the function succeeds, it returns the absolute value of the targeted polygon pan.
tilt	If the function succeeds, it returns the absolute value of the targeted polygon tilt.
zoom	If the function succeeds, it returns the absolute value of the targeted polygon zoom.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED
The library, input or output parameters are not currently initialized.
IMV_Defs::E_ERR_PARAMINVALID
The input parameters are invalid
IMV_Defs::E_ERR_OK
The function succeeded.

Remarks

- Video-in and video-out buffers must be initialized (!= Null) and allocated (size = width*height*Color_depth). Their data must respect the IMV_Buffer structure requirement.

To avoid visualisation misunderstanding, the zoom coordinates max value is smaller than 140°.

GetPositionFromInputVideoPolygon

```
unsigned long GetPositionFromInputVideoPolygon(
    int nbPts,
    int /*in*/ *xInputVideo,
    int /*in*/ *yInputVideo,
    int widthDestinationViewer,
    int heighDestinationViewer,
    float /*out*/ *pan,
    float /*out*/ *tilt,
    float /*out*/ *zoom);
```

Description

This function computes the position (pan, tilt, zoom) of a selected polygon in the input buffer. The position could be used in the SetPosition function to center another virtual camera on the area delimited by the polygon. The zoom is computed using the target polygon size and the width and height of the destination viewer to zoom the virtual camera of the destination viewer only on the area delimited by the polygon.

Parameters

nbPts	Size of xInputVideo and yInputVideo
xInputVideo	Array of X coordinates of each polygon vertex
yInputVideo	Array of Y coordinates of each polygon vertex
widthDestinationViewer	Width of the destination viewer in pixels
heightDestinationViewer	Height of the destination viewer in pixels
pan	If the function succeeds, it returns the absolute value of the targeted polygon pan.
tilt	If the function succeeds, it returns the absolute value of the targeted polygon tilt.
zoom	If the function succeeds, it returns the absolute value of the targeted polygon zoom.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED
 The library is not currently initialized.

IMV_Defs::E_ERR_PARAMINVALID
 The input parameters are invalid

IMV_Defs::E_ERR_OK
 The function succeeded.

Remarks

- Video-in buffer must be initialized (!= Null) and allocated (size = width*height*Color_depth). Its data must respect the IMV_Buffer structure requirement.

To avoid visualisation misunderstanding, the zoom coordinates max value is smaller than 140°.

GetPositionFromOuputVideoPolygon

```
unsigned long GetPositionFromOutputVideoPolygon(
    int nbPts,
    int /*in*/ *xOutputVideo,
    int /*in*/ *yOutputVideo,
    int widthDestinationViewer,
    int heighDestinationViewer,
    float /*out*/ *pan,
    float /*out*/ *tilt,
    float /*out*/ *zoom);
```

Description

This function computes the position (pan, tilt, zoom) of a selected polygon in the Output buffer. The position could be used in the SetPosition function to center another virtual camera on the area delimited by the polygon. The zoom is computed using the target polygon size and the width and height of the destination viewer to zoom the virtual camera of the destination viewer only on the area delimited by the polygon.

Parameters

nbPts	Size of xInputVideo and yInputVideo
xInputVideo	Array of X coordinates of each polygon vertex
yInputVideo	Array of Y coordinates of each polygon vertex
widthDestinationViewer	Width of the destination viewer in pixels
heightDestinationViewer	Height of the destination viewer in pixels
pan	If the function succeeds, it returns the absolute value of the targeted polygon pan.
tilt	If the function succeeds, it returns the absolute value of the targeted polygon tilt.
zoom	If the function succeeds, it returns the absolute value of the targeted polygon zoom.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_PARAMINVALID

The input parameters are invalid

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

- Video-in and video-out buffers must be initialized (!= Null) and allocated (size = width*height*Color_depth). Their data must respect the IMV_Buffer structure requirement. **To avoid visualisation misunderstanding, the zoom coordinates max value is smaller than 140°.**

GetInputVideoPointFromPosition

```
unsigned long GetInputVideoPointFromPosition(
    float pan,
    float tilt,
    int /*out*/ *xInputVideo,
    int /*out*/ *yInputVideo);
```

Description

This function computes the input buffer coordinates (*xInputVideo, yInputVideo*) of a point defined by its pan and tilt angles in the virtual camera coordinates system (see Annex 1).

Parameters

pan	Pan value of the point in the camera coordinate system*
tilt	Tilt value of the point in the camera coordinate system*
xInputVideo	If the function succeeds, it returns the X coordinate of the targeted point in the input buffer
yInputVideo	If the function succeeds, it returns the y coordinate of the targeted point in the input buffer

* see ‘annex 1’ to have a representation of the camera coordinate system

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_PARAMINVALID

The input parameters are invalid, (pan, tilt) angles doesn’t exist in the InputPicture.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

- Video-in buffer must be initialized (!= Null) and allocated (size = width*height*Color_depth). Its data must respect the IMV_Buffer structure requirement.

GetOutputVideoPointFromPosition

```
unsigned long GetOutputVideoPointFromPosition(
    float pan,
    float tilt,
    int /*out*/ *xOutputVideo,
    int /*out*/ *yOutputVideo,
    unsigned long viewIndex=1);
```

Description

This function computes the output buffer coordinates (*xOutputVideo, yOutputVideo*) of a point defined by its pan and tilt angles in the virtual camera coordinates system (see Annex 1).

Parameters

pan	Pan value of the point in the camera coordinate system*
tilt	Tilt value of the point in the camera coordinate system*
xOutputVideo	If the function succeeds, it returns the X coordinate of the targeted point in the output buffer
yOutputVideo	If the function succeeds, it returns the y coordinate of the targeted point in the output buffer
viewIndex	Index of the virtual camera for the Quad mode. See remark section to know which value corresponds to which virtual camera.

* see ‘annex 1’ to have a representation of the camera coordinate system

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_PARAMINVALID

The input parameters are invalid, (pan, tilt) angles doesn’t exist in the OutputPicture.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

- Video-in and video-out buffers must be initialized (!= Null) and allocated (size = width*height*Color_depth). Their data must respect the IMV_Buffer structure requirement.

- **Note that this function is under development and may return bad results until the final release of the Immervision Enables 2.0 library.**

- The Quad mode displays 4 Single virtual cameras simultaneously. The indexes of the virtual cameras are as follows:

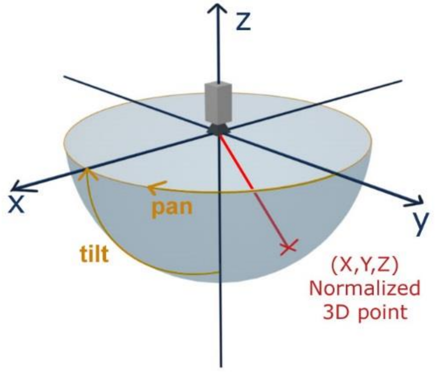
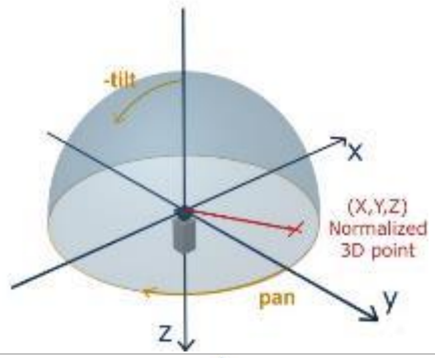
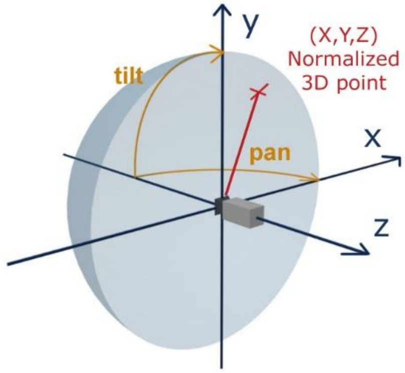
1	**2**
3	**4**

GetPositionFrom3D

```
unsigned long GetPositionFrom3D( float x,
                                 float y,
                                 float z,
                                 float /*out*/ *pan,
                                 float /*out*/ *tilt);
```

Description

This function computes the position (pan, tilt) of a 3D point in an orthonormal basis. The position could be used in the SetPosition function to center the virtual camera on the selected point.

 z x y pan tilt (X,Y,Z) Normalized 3D point	Ceiling mode
 -tilt x (X,Y,Z) Normalized 3D point pan y z	Ground mode
 y (X,Y,Z) Normalized 3D point tilt pan x z	Wall mode

Parameters

x	X coordinate in the camera coordinates system*
y	Y coordinate in the camera coordinates system*
z	Z coordinate in the camera coordinates system*
pan	If the function succeeds, it returns the absolute value of the targeted point pan.
tilt	If the function succeeds, it returns the absolute value of the targeted point tilt.

* see ‘annex 1’ to have a representation of the camera coordinates system

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_PARAMINVALID

The input parameters are invalid. The 3D point must not be visible in the picture.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks

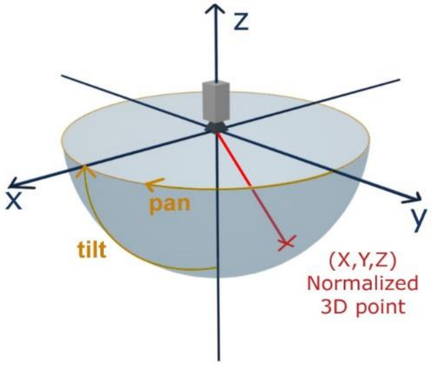
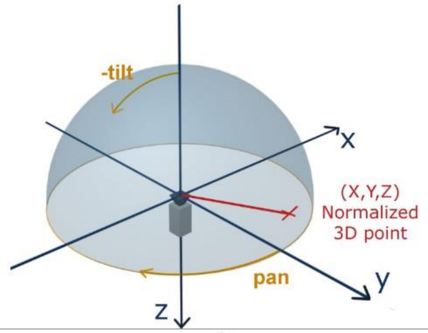
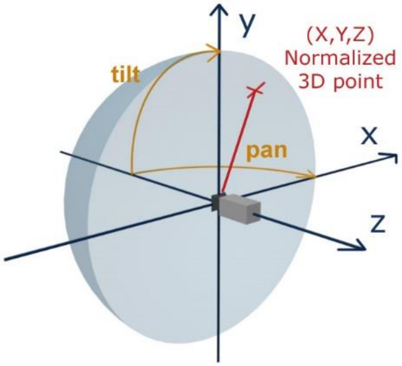
- Video-in and video-out buffers must be initialized (!= Null) and allocated (size = width*height*Color_depth). Their data must respect the IMV_Buffer structure requirement.

Get3DFromPosition

```
unsigned long Get3DFromPosition( float pan,
                                 float tilt,
                                 float /*out*/ *x,
                                 float /*out*/ *y,
                                 float /*out*/ *z);
```

Description

This function computes the 3D normalized coordinates (x,y,z) of a position (pan,tilt). The 3D coordinates could be used to locate a target in the real world.

 z x y pan tilt (X,Y,Z) Normalized 3D point	Ceiling mode
 -tilt x (X,Y,Z) Normalized 3D point pan y z	Ground mode
 y (X,Y,Z) Normalized 3D point tilt pan x z	Wall mode

Parameters

pan	Absolute value of pan in the coordinates system*
tilt	Absolute value of tilt in the coordinates system*
x	X coordinate in the camera coordinates system*
y	Y coordinate in the camera coordinates system*
z	Z coordinate in the camera coordinates system*

* see ‘annex 1’ to have a representation of the camera coordinates system

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_PARAMINVALID

The input parameters are invalid. The position must not be visible in the picture.

IMV_Defs::E_ERR_OK

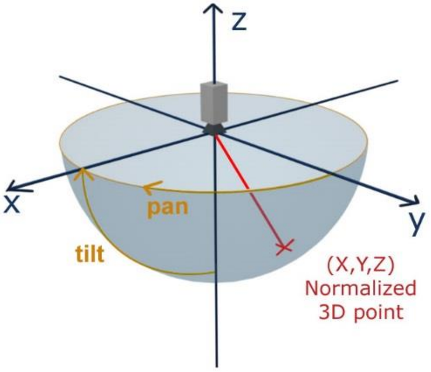
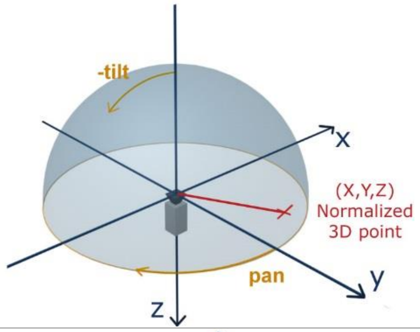
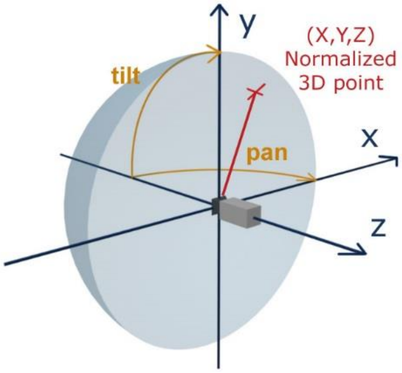
The function succeeded.

Remarks

- Video-in and video-out buffers must be initialized (!= Null) and allocated (size = width*height*Color_depth). Their data must respect the IMV_Buffer structure requirement.

Annex 1

The camera coordinates system

 z x y pan tilt (X,Y,Z) Normalized 3D point	Ceiling mode
 -tilt x (X,Y,Z) Normalized 3D point pan y z	Ground mode
 y (X,Y,Z) Normalized 3D point tilt pan x z	Wall mode

Remarks

(pan,tilt) orientations are adapted to the 360° navigation referential.

(x,y,z) coordinate system stays the same.

IMV_CAMERAFLATSURFACEINTERFACE - FUNCTIONS

GetFlatSurfaceModel

```
unsigned long GetFlatSurfaceModel(
                                   int flatSurfaceIndex,
                                   int /*out*/ *nbVertex2D,
                                   Vertex2D /*out*/ **vertex2DCoordinates,
                                   Vertex2D /*out*/ **vertex2DTxCoords);
```

Description

This function gets the 2D FlatSurface model of the output to be displayed by a GPU engine (OpenGl, DirectX, etc.).

The 2D FlatSurface model is represented by 2 arrays:

- *vertex2DCoordinates represents the coordinates of each point of the* FlatSurface*, organized as a triangle list. Points position depends of the output buffer dimension and the* 360° navigation referential*.*
- *vertex2D*TxCoords represents the coordinates of each point on the source picture.

Note: Vertex2DCoordinates and Vertex2DTxCoords are two arrays of equal dimensions.
To each vertex corresponds texture coordinates.
Ex: The point Vertex2DCoordinates[**pt**] is represented on the source picture at the position Vertex2DTxCoords[**pt**].

Parameters

flatSurfaceIndex	The index of the FlatSurface to get. Depending on the view type (Single, Quad, Perimeter), several FlatSurfaces can be exported. The Single view exports 1 FlatSurface. The Quad view exports 4 FlatSurfaces. The perimeter view: - exports 2 FlatSurfaces when the 360° navigation referential is set to ground or ceiling - exports 1 FlatSurface when the 360° navigation referential is set to wall The perimeter custom view exports 1 FlatSurface.
nbVertex2D	Number of vertices used in the both 2D shape (Vertex2DCoordinates and Vertex2DTxCoords).
Vertex2DCoordinates	Vertex2D array containing the 2d coordinates of the FlatSurface.
Vertex2DTxCoords	Vertex2D array of the source picture texture coordinates corresponding to the FlatSurface defined by Vertex2DIndices.

Return codes

IMV_Defs::E_ERR_NOTINITALIZED

The library is not currently initialized.

IMV_Defs::E_ERR_OK

The function succeeded.

Remarks -