Hantro

Application Programming Interface 1.0

7280 Video Stabilization

User Manual Version 1.0

Copyright Information

Hantro Products Oy
Nahkatehtaankatu 2
90100 Oulu
FINLAND
www.hantro.com

Glossary

API	Application Programming Interface
VGA	Video Graphics Array (resolution 640x480 pixels)
VS	Video Stabilization

Table of Contents

1 Introduction

This document is aimed to represent the Application Programming Interface (API) of the Hantro 7280 Video Stabilization block. The product is able to reduce the unwanted motion in a video sequence thus increasing the quality and compression efficiency of the video encoding. The proprietary algorithm used for detecting the global motion is highly resistant to noise and brightness changes in the camera frame and is also computationally inexpensive.

The usage of the API is described in chapter 2. The detailed function interfaces and data types are introduced in chapter 4. Chapter 5 gives application examples. References are introduced in the end of the document.

In the document all functions, parameters, data types and code are described in `Courier new (syntax style)` font. Notes and filenames are written in *italic* expression.

This document assumes that the reader understands the basics of video processing and frame formats.

2 Video Stabilization Usage

The video stabilization is operated via the Application Programming Interface (API). The API usage can be seen in the sequence chart FIGURE 1.

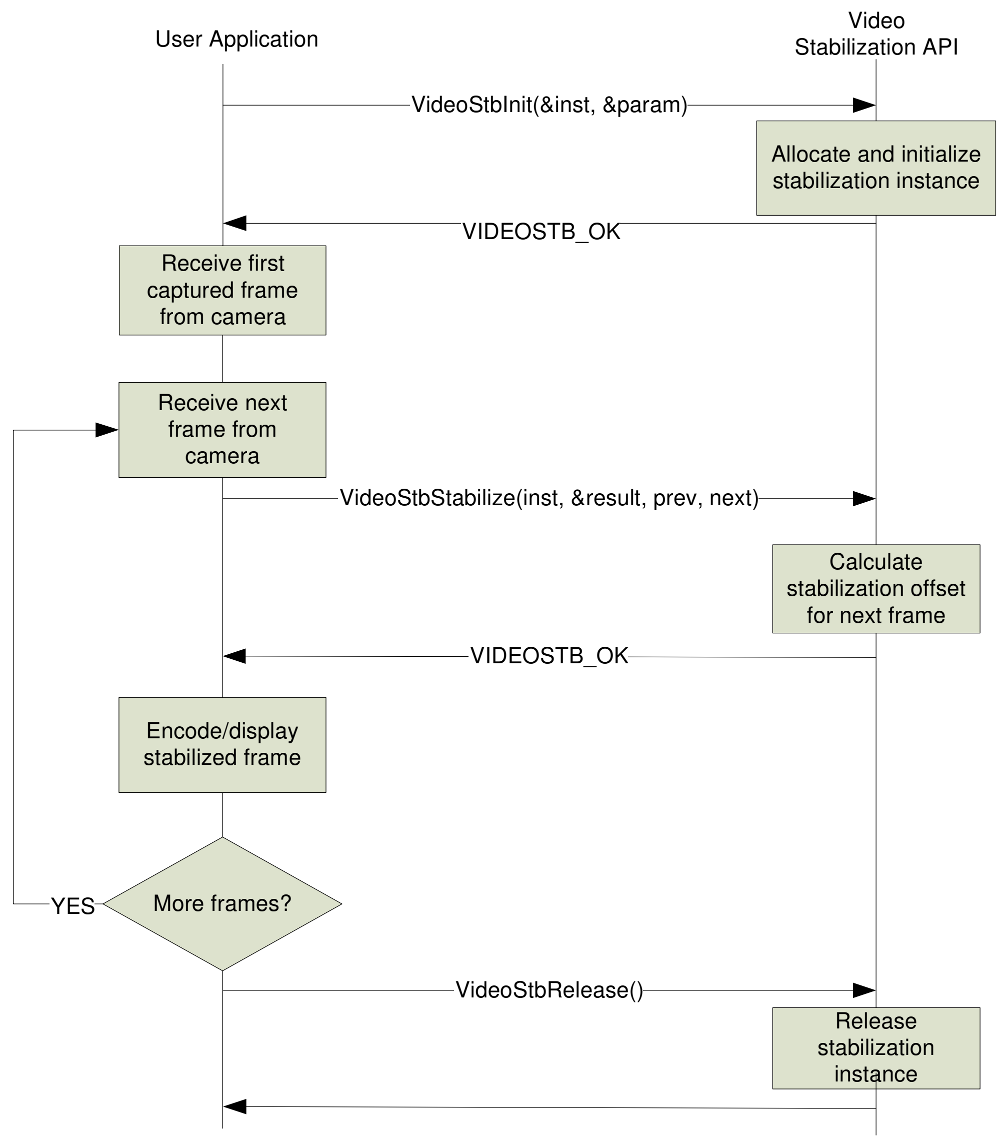
User Application
Video Stabilization API
VideoStbInit(&inst, &param)
Allocate and initialize stabilization instance
VIDEOSTB_OK
Receive first captured frame from camera
Receive next frame from camera
VideoStbStabilize(inst, &result, prev, next)
Calculate stabilization offset for next frame
VIDEOSTB_OK
Encode/display stabilized frame
More frames?
YES
VideoStbRelease()
Release stabilization instance

FIGURE 1. VIDEO STABILIZATION CALLING SEQUENCE

2.1 Initialization of the video stabilization

The video stabilization procedure is started with the `VideoStbInit()` function, which allocates memories, initializes the stabilization, and sets all the parameters to a default state. The very first frame is set to be in the middle of the input frame. If the initialization is successful the function returns `VIDEOSTB_OK` and stores a reference to the stabilization instance in the parameter `inst`. The place where the stabilization instance is stored must be reserved by the user and it will be used as an identifier for subsequent stabilization API function calls. Multiple instances of the stabilization can exist simultaneously.

2.2 Stabilizing video frames

All the actual stabilization is done by the `VideoStbStabilize()` function. This function takes the following input parameters: the stabilization instance, a pointer to an output structure, base address of the previous camera frame and base address of the current camera frame.

The output structure is used for returning data from the video stabilization. The structure holds the horizontal and vertical stabilization offsets that point the stabilized position in the current frame. FIGURE 2 shows the relationship of the frame dimensions used by stabilization and demonstrates the effect of stabilizing a video frame. Frame 0 is the first frame and the stabilized frame is positioned in the middle of the camera frame. Frame 1 has been stabilized and the stabilized frame has moved 4 pixels left. The edges around the stabilized frame (shown in dark) have to be cropped out when encoding or displaying the video thus creating a more stable video.

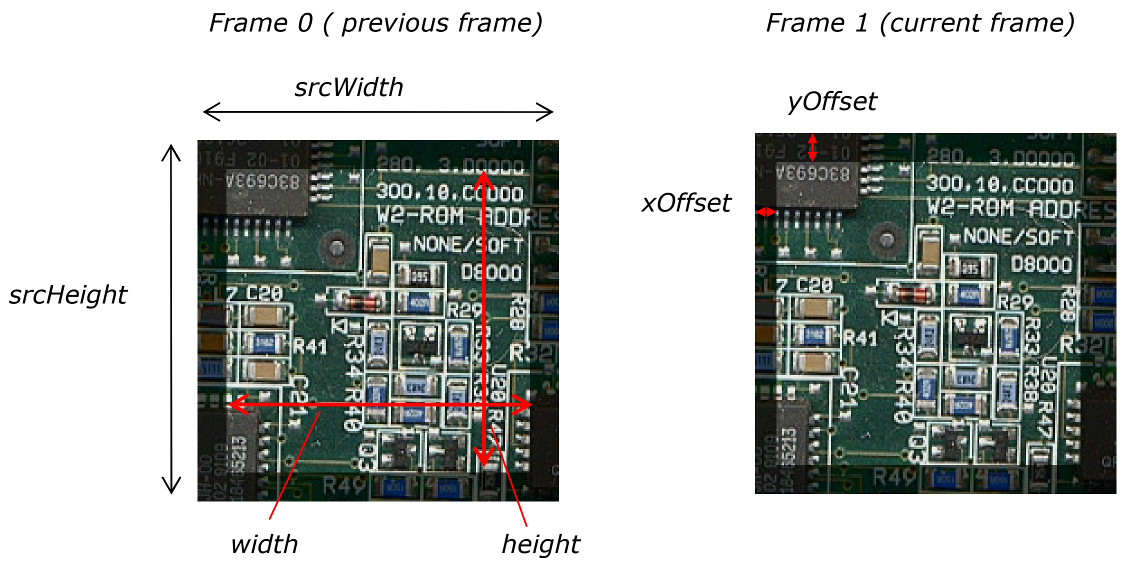

FIGURE 2. FRAME DIMENSIONS USED IN STABILIZATION

The input frame for the video stabilization must be YCbCr (4:2:0 or 4:2:2) format. The video stabilization only uses the luminance samples of the input frames.

3 Frame Storage Format

The input picture format of the encoder is planar YCbCr 4:2:0, semiplanar YCbCr 4:2:0 or interleaved YCbCr 4:2:2. Figure 3 describes how the planar YCbCr 4:2:0 picture is stored in external memory. The luminance and both chrominance components are stored in three buffers and they must be located in a linear and physically contiguous memory block.

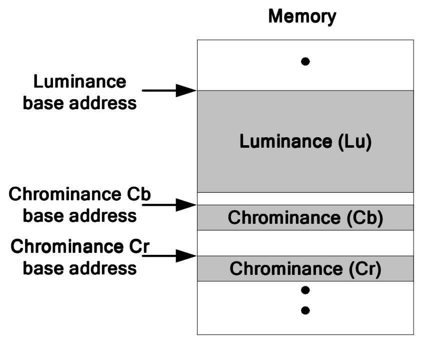

FIGURE 3. PLANAR YCBCR 4:2:0 PICTURE STORAGE IN EXTERNAL MEMORY

Figure 4 describes the format of semiplanar YCbCr 4:2:0 picture. The luminance is equal to planar format but the Cb and Cr chrominance components are interleaved together.

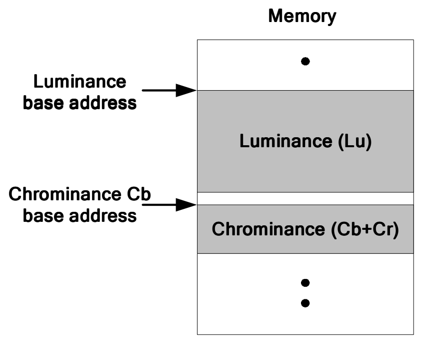

FIGURE 4. SEMIPLANAR YCBCR 4:2:0 PICTURE STORAGE IN EXTERNAL MEMORY

Figure 5 describes the format of interleaved YCbCr 4:2:2 picture where all the three components are interleaved. The sample order is selected at initialization to be either Y-Cb-Y-Cr or Cb-Y-Cr-Y.

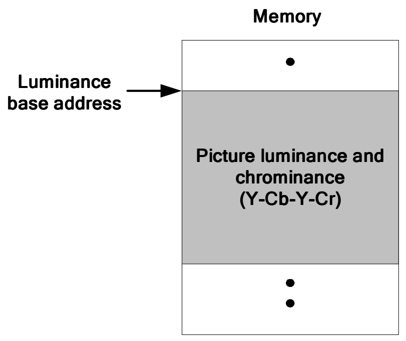

FIGURE 5. INTERLEAVED YCBCR 4:2:2 PICTURE STORAGE IN EXTERNAL MEMORY

The input picture data endianess is set when the encoder software is compiled and can't be changed during run-time. The sizes of the different components are based on the picture's dimensions and the input format:

Planar YCbCr 4:2:0	Y = width x height [bytes]
	Cb = width/2 x height/2 [bytes]
	Cr = width/2 x height/2 [bytes]

Semiplanar YCbCr 4:2:0	Y = width x height [bytes] Cb+Cr = width x height/2 [bytes]
Interleaved YCbCr 4:2:2	Y+Cr+Cr = width x height x 2 [bytes]

3.1 Frame size limitations

YCbCr 4:2:0 limitations regarding the input frame size and the stabilized frame size:

stabilized_width % 4 = 0
stabilized_height % 2 = 0
input_width >= stabilized_width + 8
input_height >= stabilized_height + 8
input_width >= (stabilized_width + 15) & (~15)
input_stride % 8 = 0

YCbCr 4:2:2 limitations regarding the input frame size and the stabilized frame size:

stabilized_width % 4 = 0
stabilized_height % 2 = 0
input_width >= stabilized_width + 8
input_height >= stabilized_height + 8
input_width >= (stabilized_width + 15) & (~15)
input_stride % 8 = 0

Where:

%	- remainder operator
&	- bitwise AND operator
~	- bitwise NOT operator

4 Interface Functions

This chapter describes the API declared in vidstbapi.*h*.

Common numeric data types are declared in *basetype.h*:
`u8` – unsigned 8 bits integer value
`i8` – signed 8 bits integer value
`u16` – unsigned 16 bits integer value
`i16` – signed 16 bits integer value
`u32` – unsigned 32 bits value
`i32` – signed 32 bits value

4.1 VideoStbGetApiVersion

Syntax

```
VideoStbApiVersion VideoStbGetApiVersion(void)
```

Purpose

Returns API version information of the video stabilization. Does not require a prior initialization.

Parameters

None.

Return value

`VideoStbApiVersion`
A structure containing the API's major and minor version number.

Description

```
typedef struct
{
    u32 major;
    u32 minor;
} VideoStbApiVersion;
```

`major`
The major version number.

`minor`
The minor version number.

4.2 VideoStbGetBuild

Syntax

```
VideoStbBuild VideoStbGetBuild(void)
```

Purpose

Returns the stabilization’s hardware and software build information. Does not require a prior initialization.

Parameters

None.

Return value

VideoStbBuild

A structure containing the video stabilization’s build information.

```
typedef struct
{
    u32 swBuild;
    u32 hwBuild;
} VideoStbBuild;
```

`swBuild`

The internal release version of the 7280 video stabilization software.

`hwBuild`

The internal release version of the 7280 hardware.

4.3 VideoStbInit

Syntax

```
VideoStbRet VideoStbInit(VideoStbInst *instAddr,
                         const VideoStbParam *param)
```

Purpose

Initializes and returns new instance of the video stabilizer.

Parameters

VideoStbInst `*instAddr`

Points to a space where the new stabilization instance will be stored.

VideoStbParam **param*

Pointer to the structure that holds the parameters for configuring the instance.

Return value

VIDEOSTB_OK – initialization successful
VIDEOSTB_NULL_ARGUMENT – error, a pointer argument had an invalid NULL value.
VIDEOSTB_INVALID_ARGUMENT – error, one of the arguments was invalid.
VIDEOSTB_MEMORY_ERROR – error, the stabilization was not able to allocate memory
VIDEOSTB_EWL_ERROR – error, the stabilization system interface failed to initialize

Description

```
typedef struct
{
    u32 inputWidth;
    u32 inputHeight;
    u32 stride;
    u32 stabilizedWidth;
    u32 stabilizedHeight;
    VideoStbYuvFormat format;
} VideoStbParam;
```

`inputWidth`
The width of the camera input frame in pixels.
Valid values: [104, 1920]

`inputHeight`
The height of the camera input frame in pixels.
Valid values: [104, 1920]

`stride`
The camera input frame horizontal scanline in pixels. Has to be an 8 multiple.
Valid values: [104, 1920]

`stabilizedWidth`
The width of the stabilized frame in pixels. Must be a multiple of 4.
Valid values: [96, inputWidth-8].

`stabilizedHeight`
The height of the stabilized frame in pixels. Must be a multiple of 2.
Valid values: [96, inputWidth-8].

`format`
Specifies the input YUV frame format. The formats are described in Video Frame Storage Format chapter. The possible values are:
VIDEOSTB_YUV420_PLANAR
VIDEOSTB_YUV420_SEMIPLANAR
VIDEOSTB_YUV422_INTERLEAVED_YUYV
VIDEOSTB_YUV422_INTERLEAVED_UYVY

The formats VIDEOSTB_YUV420_PLANAR and VIDEOSTB_YUV420_SEMIPLANAR have the same way of storing the luminance data samples, which is needed by the stabilizer. In this sense the stabilizer does not differentiate between them.

Figure 6 describes the relation between the different frame parameters and the organization of frame data in the memory.

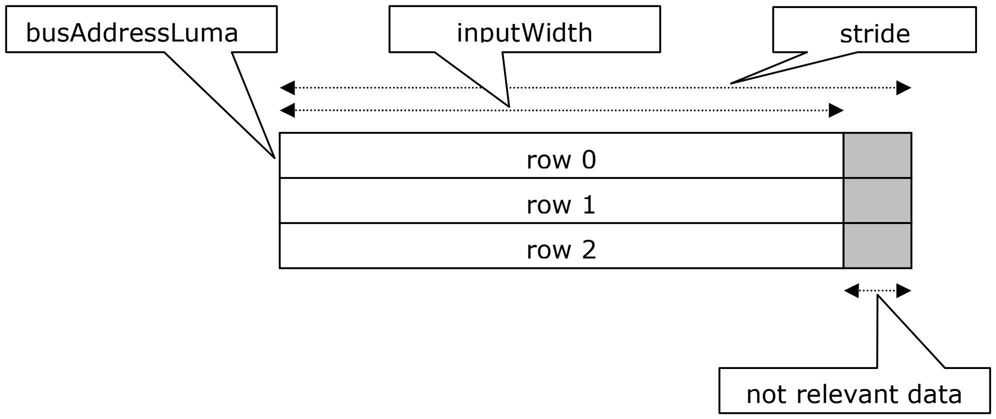

FIGURE 6 FRAME DATA ORGANIZATION IN MEMORY

4.4 VideoStbRelease

Syntax

VideoStbRet `VideoStbRelease(`**VideoStbInst** `vidStab)`

Purpose

Releases a video stabilization instance. This will free all the resources allocated at the initialization phase.

Parameters

VideoStbInst `vidStab`

The stabilization instance to be released. This instance was created earlier with a `VideoStbInit` call.

Return value

`VIDEOSTB_OK` – release successful
`VIDEOSTB_NULL_ARGUMENT` – Error, a pointer argument had an invalid NULL value
`VIDEOSTB_INSTANCE_ERROR` – Error, the instance to be released is invalid.

4.5 VideoStbReset

Syntax

```
VideoStbRet VideoStbReset(VideoStbInst vidStab,
                            const VideoStbParam *param)
```

Purpose

Resets and reconfigures an existing stabilization instance. It can be used when an earlier allocated, in use stabilization instance needs to be reset to the initial state for example in case of scene change or beginning of a new video sequence.

Parameters

VideoStbInst vidStab

The video stabilization instance in use. This instance was created earlier with a `VideoStbInit` call.

VideoStbParam **param*

Pointer to the structure that holds the parameters for configuring the instance. See `VideoStbInit` for description.

Return value

`VIDEOSTB_OK` – successful call.
`VIDEOSTB_NULL_ARGUMENT` – Error, wrong NULL argument.
`VIDEOSTB_INSTANCE_ERROR` – Error, the instance provided is not valid.
`VIDEOSTB_INVALID_ARGUMENT` – error, one of the arguments was invalid.

4.6 VideoStbStabilize

Syntax

```
VideoStbRet VideoStbStabilize(VideoStbInst vidStab,
                              VideoStbResult * result,
                              u32 referenceFrameLum,
                              u32 stabilizedFrameLum)
```

Purpose

Stabilizes the motion in a new frame based on a previously stabilized frame. This function has to be called once for each video frame that needs to be stabilized. The algorithm compares the new frame to the previous frame to find the global motion. The global motion is then filtered using an adaptive motion filter to separate any unwanted motion (camera shaking) from the intentional motion (camera panning). The resulting motion is compensated by moving the stabilized frame inside the captured frame. The position of the stabilized frame is returned as an offset from the top-left corner of the camera frame.

Parameters

VideoStbInst `vidStab`
The video stabilization instance in use.

VideoStbResult `*result`
Pointer to the structure where the stabilized frame position will be stored.

u32 `referenceFrameLum`
Bus address of the reference input frame's luminance data. During the stabilization process this is the previously stabilized frame.

u32 `stabilizedFrameLum`
Bus address of luminance data of the next frame to be stabilized.

Return value

`VIDEOSTB_OK` - API function returned successfully.
`VIDEOSTB_NULL_ARGUMENT` - Pointer argument is NULL.
`VIDEOSTB_INVALID_ARGUMENT` – Parameter value is invalid.
`VIDEOSTB_INSTANCE_ERROR` – Error, the instance provided is not valid.
`VIDEOSTB_HW_RESERVED` – error, could not reserve hardware resources. Try again.
`VIDEOSTB_HW_TIMEOUT` **–** error, the wait for a hardware finish has timed out. Stabilization not done.
`VIDEOSTB_HW_BUS_ERROR` **–** error. This can be caused by invalid bus addresses, addresses that push the hardware to access an invalid memory area. Stabilization not done.
`VIDEOSTB_HW_RESET` – error. Hardware was reset by external means. Stabilization not done.
`VIDEOSTB_SYSTEM_ERROR` **–** error, a fatal system error was caught. Stabilization not done. The stabilization instance has to be released.

Description

```
typedef struct
{
    u32 stabOffsetX;
    u32 stabOffsetY;
} VideoStbResult;
```

`stabOffsetX`
Horizontal offset in pixels from top-left corner of current frame to top-left corner of stabilized frame.
`stabOffsetY`
Vertical offset in pixels from top-left corner of current frame to top-left corner of stabilized frame.

The starting point of the stabilization (after initialization or reset) is considered to be the middle position of the input frame. In a normal operation mode after a successful return, the stabilized frame will serve as reference for processing another frame (`referenceFrameLum = stabilizedFrameLum` and `stabilizedFrameLum` will have to be updated).

5 Application Example

The following example codes demonstrate the basic usage of the video stabilizer. The code uses the API functions and external functions `ReadYuv420PlanarFrame()`, `AllocateDMABuffer()` and `FreeDMABuffer()`. The examples are simplified and there is no error handling. *`standalone.c`* is a more thorough example on how to use the stabilizer.

The source code given below is not guaranteed to compile as such.

5.1 VGA video stabilization

Following example stabilizes a VGA resolution video when the captured video has a resolution bigger with 16 pixels.

```
#include "basetype.h"
#include "vidstbapi.h"

VideoStbInst videoStab;
VideoStbParam stabParam;
VideoStbRet ret;

u32 frameBusAddress;
u32 nextFrameBusAddress;

i32 last = 0;

/* Allocate linear memory resources. This implementation
   is OS specific and not part of this example. */
AllocDMAMemory(&frameBusAddress);
AllocDMAMemory(&nextFrameBusAddress);

stabParam.format = VIDEOSTB_YUV420_PLANAR;

/* VGA size stabilized video */
stabParam.stabilizedWidth = 640;
stabParam.stabilizedHeight = 480;

/* Captured size from camera is 16 pixels bigger */
stabParam.inputWidth = stabParam.stabilizedWidth + 16;
stabParam.inputHeight = stabParam.stabilizedHeight + 16;

/* stride equals width, no extra padding */
stabParam.stride = stabParam.inputWidth;

if((ret = VideoStbInit(&videoStab, &stabParam)) != VIDEOSTB_OK)
{
    printf("VideoStbInit ERROR: %d\n", ret);
    goto end;
}

ReadYuv420PlanarFrame(frameBusAddress);    /* read first frame */
```

```
while(!last)
{
    VideoStbResult result;

    /* read next frame to be stabilized */
    ReadYuv420PlanarFrame(nextFrameBusAddress);

    /* stabilize next */
    ret = VideoStbStabilize(videoStab,
                            &result,
                            frameBusAddress,
                            nextFrameBusAddress);

    if(ret != VIDEOSTB_OK)
    {
        printf("VideoStbStabilize ERROR: %d\n", ret);
        break;
    }

    printf("Stabilization result (%2d,%2d)\n",
                                              result.stabOffsetX,
                                              result.stabOffsetY);

    /* stabilized frame is becoming reference now (swap buffers) */
    {
        u32 tmp = frameBusAddress;
        frameBusAddress = nextFrameBusAddress;
        nextFrameBusAddress = tmp;
    }

}

/* release stabilization instance */
VideoStbRelease(videoStab);

FreeDMAMemory(frameBusAddress);
FreeDMAMemory(nextFrameBusAddress);

end:
```

References

VISIBLY BETTER

Headquarters:
Hantro Products Oy
Kiviharjunlenkki 1, FI-90220 Oulu, Finland
Tel: +358 207 425100, Fax: +358 207 425299

Hantro Finland	Hantro Germany	Hantro USA	Hantro Japan	Hantro Taiwan	Hantro Korea
Lars Sonckin kaari 14	Ismaninger Str. 17-19	1762 Technology Drive	Yurakucho Building 11F	6F, No. 331	#518 ho, Dongburoot, 16-2
FI-02600	81675	Suite 202. San Jose	1-10-1, Yurakucho	Fu-Hsing N. Rd.	Sunaedong, Bundanggu,
Espoo	Munich	CA 95110,	Chiyoda-ku, Tokyo	Taipei	Seongnamsi, Kyunggido,
Finland	Germany	USA	100-0006, Japan	Taiwan	463-825 Korea
Tel: +358 207 425100	Tel: +49 89 4130 0645	Tel: +1 408 451 9170	Tel: +81 3 5219 3638	Tel: +886 2 2717 2092	Tel: +82 31 718 2506
Fax: +358 207 425298	Fax: +49 89 4130 0663	Fax: +1 408 451 9672	Fax: +81 3 5219 3639	Fax: +886 2 2717 2097	Fax: +82 31 718 2505

Email: sales@hantro.com
WWW.HANTRO.COM