CvCam

Reference manual

Contents

Introduction

CvCam is a universal cross-platform module for processing video stream from digital video cameras. It is implemented as a dynamic link library (DLL) for Windows and as a shared object library (so) for linux systems and provides a simple and convenient Application Programming Interface (API) for reading and controlling video stream, processing its frames and rendering the results. CvCam is distributed as a part of Intel's OpenCV project under the same license and uses some functionality of the Open Source Computer Vision Library.

Common Usage

The simplest way to use CvCam from a C++ program for a single video source is shown at the following listing:

```
#include “cvcam.h”
void callback(IplImage* image);
int main()
{
        int ncams = cvcamGetCamerascount( );//returns the number of available cameras in the
                                                //system
        cvcamSetProperty(0, CVCAM_PROP_ENABLE, &cvcamone); //Selects the 1-st found
                                                                //camera
        cvcamSetProperty(0, CVCAM_PROP_RENDER, &cvcamone)  //We’ll render stream
                                                                //from this source
        //here I assume your create a window and store it’s id in MyWin variable.
        //MyWin is of type HWND on Windows and Window on linux

        cvcamSetProperty(0, CVCAM_PROP_WINDOW, &MyWin);   // Selects a window for
                                                                //video rendering
        cvcamSetProperty(0, CVCAM_PROP_CALLBACK, callback);//this callback will
                                                                //process every frame
        cvcamInit( );
        cvcamStart( );
        //Your app is working
        cvcamStop( );
        cvcamExit( );
        return 0;
}
void callback(IplImage* image)//Draws blue horizontal lines across the image
{
IplImage* image1 = image;
int i,j;

assert (image);

for(i=0; i<image1->height; i+=10)
{
  for(j=(image1->widthStep)*i; j<(image1->widthStep)*(i+1);
        j+=image1->nChannels)
  {
     image1->imageData[j]   = (char)255;
     image1->imageData[j+1] = 0;
     image1->imageData[j+2] = 0;


  }

}

}
```

The first function to be called is cvcamGetCamerasCount( ); It prepares library and returns the number of available cameras at the system. On Linux it’s also important to call it before your initialize the X window system as it prepares X to work with linux threads. Your can also call XInitThreads( ) at the beginning of your app instead.

The next step is to choose a camera and set it up. cvcamSetProperty(int camera, char* property, void* arg) can do this for you. See **CvCam Properties Interface** for complete list of properties and arguments. The first argument for this function is the number of the camera for which the property is set, the second is a property's name and the third is a pointer to some data, which actual type depends on the property. In the example above we selected a camera and told CvCam to render it into the window we specified.

The third optional step is frame processing with a callback. You can provide a function, which will be called on every frame and can change it's data. It should be of type void and takes a pointer to a IplImage structure, declared in "ipl.h" header file. This is done by CVCAM_PROP_CALLBACK property and the third argument to cvcamSetProperty is a pointer to your callback function;

Next step is calling cvcamInit( ), after which everything is ready for streaming;

cvcamStart( ) just starts the stream. It won't block your app and returns immediately. The stream continues until cvcamStop( ) is called.

And the last is cvcamExit( ) –Frees all resources used by CvCam.

Howto:

Begin work with cvcam

Call cvcamGetCamerasCount(). This not only returns the number of cameras in the system, but also prepares cvcam. Under linux be sure to do this call before initializing X, or to call XInitThreads() before it.

Select a camera

You can select single or multiple cameras in 2 ways. The first is using a camera selection dialog with cvcamSelectCamera. See an example below:

```
//Prototype
/*Pops up a camera(s) selection dialog
Return value - number of cameras selected (0,1 or 2);
Argument: an array of selected cameras numbers
NULL if none selected. Should be released with free() when not needed.
if NULL passed, not used.
*/
CVCAM_API int cvcamSelectCamera(int** out);

FunctionThatSelectsCamera()
{
        int* out;
        int nselected = cvcamSelectCamera(&out);
        if(nselected>0)
                printf("the 1-st selected camera is camera number %d", out[0]);
        if(nselected == 2)
                printf("the 2-nd selected camera is camera number %d", out[1]);

        free(out);
        return;


}
```

Note: if you don't need selected cameras numbers, simply call cvcamSelectCamera(NULL)
Note2: Linux version of cvcam currently has no implementation of cvcamSelectCamera.

The second, non-dialog way is to use CVCAM_PROP_ENABLE property like this:

```
int desiredcamera = 0;//for example
cvcamSetProperty(desiredcamera, CVCAM_PROP_ENABLE,CVCAMTRUE);
```

Set up camera(s)

See **CvCam Properties Interface.**

Render the video stream

1. Tell cvcam to render your camera:

```
cvcamSetProperty(YourCamera, CVCAM_PROP_RENDER, CVCAMTRUE)
```

2. Select the window

```
cvcamWindow mywin = …;//here I mean your initialize mywin with some
                                        // existing window id/
cvcamSetProperty(YourCamera, CVCAM_PROP_WINDOW, &mywin);
```

3.Set output width and height of the video.

```
int width = 320;
int height = 240;
cvcamSetProperty(YourCamera, CVCAM_PROP_RNDWIDTH, &width);
cvcamSetProperty(YourCamera, CVCAM_PROP_RNDHEIGHT, &height);
```

Note: if your pass NULL as a window, or don't set it, it'll be created under Windows systems
Note 2: If you don't set width and height those of the source are used.
Note 3 : cvcamWindow has type HWND under Windows and type Window under linux.

Make the settings active

Call cvcamInit();

Control the video

Call cvcamStart();
cvcamStop();
cvcamPause();
cvcamResume();
They take no arguments and return 0 on success.

Display cameras property pages

To display the property page for camera YourCamera call
cvcamGetProperty(YourCamera, CVCAM_CAMERAPROPS, NULL);

To display the video format property page for the same camera call
cvcamGetProperty(YourCamera, CVCAM_VIDEOFORMAT, NULL);

Note: a camera may have no one or both property pages under Windows and the content of the pages depends on the particular driver.
Note2: Property pages are displayed only for enabled camera(s) and after a call to cvcamInit() under Windows.

Process video frames

You can set your own callback function, which will process every captured frame and change it's data. It can be done with CVCAM_PROP_CALLBACK property.

```
void callback(IplImage* image)
{
	//Do everything you want with the image
}
cvcamSetProperty(0, CVCAM_PROP_CALLBACK, callback)
```

Note:under linux callback currently is called on every frame being rendered or got with "raw_image" property.
Under windows –on every frame captured.

Get the pointer to last frame

```
//Example for camera 0
	IplImage* image;
	CvcamPause();
	cvcamGetProperty(0,"raw_image",&image)
	//Do what you want with the image
	cvcamResume();
```

Note: you don't need to free the data pointed to by image.
Note2: You should use cvcamPause and cvcamResume to prevent changes of the data.
Note3. This isn't currently implemented under Windows

Play an avi file and process it's frames if needed

Use cvcamPlayAVI(const char* file,
void* window,
int width,
int height,
void* callback)

The default call cvcamPlayAVI(0,0,0,0,0) opens an "open file" dialog and if a file was chosen creates a video window and plays the file into it without any procession. If an existing file name is passed as a first argument this file is played. The second argument is a window id to play video into it (HWND or Window depending on platform)
Width and Height specify the size of the output video window. If 0 passed, those that are set in the file are used. Callback is quiet as for processing video from camera – void callback(IplImage* image).

Example(C++ for Windows):

```
extern "C"
{
	#include "highgui.h"
}
int main()
{
	char* filename = "C:\cartoon.avi";

	named_window("cvcam window", 0);
	HWND mywin = get_hwnd_byname("cvcam window");
	cvcamPlayAVI( filename, mywin, 320, 240, mycallback);
	return 0;
}

void mycallback(IplImage* img) //Paints top half of image black
{
	memset(img->imageData, 0, img->imageSize/2);

}
```

Note: This function blocks the calling thread until the file finishes playing.
Note: this function isn't implemented under Linux yet.

Work with multiple cameras under linux

Selecting multiple cameras under linux with setting CVCAM_PROP_ENABLE property will make them all to work independently

Work with multiple cameras under Windows

Enabling two cameras under Windows results in activating cvSyncFilter, which synchronizes the streams and enables setting stereo callback for 2 synchronous frames. Set it as below:

```
void stereocallback(IplImage* image1, IplImage* image2);
cvcamSetProperty(0, CVCAM_STEREO_CALLBACK , stereocallback);
```

Note: single callbacks for each camera are not active in this mode currently;

Finish working with cvcam

Call cvcamExit();

Cvcam Reference
cvcam API

cvcamExit

Declaration:
int cvcamExit();

Parameters:
None.

Return Value:
Always 0.

Remarks:
Frees all resources, used by cvcam.

cvcamGetCamerasCount

Declaration:
int cvcamGetCamerasCount();

Parameters:
None.

Return Value:
Number of found cameras.

Remarks:
The first procedure to call in most every cvcam based program. Initializes the cvcam library. Under linux initializes X window system to work with multiple threads also.

cvcamGetProperty

Declaration:
cvcamGetProperty(int camera, const char* property, void* value);

Parameters:
int **camera** (in) – a number of the camera in 0-based index of cameras found in the system.

const char* **property** (in) – a name of the property (See cvcam Properties Interface for details).

void* **value** (out) – depends on the property's name. (See cvcam Properties Interface for details).

Return Value:
0 on success, negative error code for error.

Remarks:
If successful, **value** will contain the value of the specified **property** for the specified **camera**. (See cvcam Properties Interface for details).

cvcamInit

Declaration:
```
int cvcamInit();
```

Parameters:
None.

Return Value:
1 on success, 0 on error.

Remarks:
This function makes the settings, set by cvcamSetProperty active and does the final step of cvcam initialization. Don't be confused – it's not the first function, you call. It assumes that cvcamGetCamerasCount has already been called and usually that some properties have been set.

cvcamPause

Declaration:
```
int cvcamPause();
```

Parameters:
None.

Return Value:
Always 0.

Remarks:
Just pauses the video stream if it is running.

cvcamPlayAVI

Declaration:

```
int cvcamPlayAVI(const char* file,
                 void* window,
                 int width,
                 int height,
                 void* callback)
```

Parameters:

const char* **file** (in) – the name of an existing file on disk or NULL .

void* **window** (in) – an existing HWND (Windows systems) or Window(Linux systems) or NULL.

int **width** (in) – width of the displayed video;

int **height** (in) – height of the displayed video;

void* **callback** (in) – a pointer to a callback function void callback(IplImage* img), which will process every frame before displaying it.

Return Value:

0 on success, -1 on eror.

Remarks:

Plays the **file** of type .avi into a specified **window**. The output video size is set to **width** x **height.** Every frame is processed with a **callback** before displaying.

If the **file** parameter is NULL, the "file open" dialog is displayed.

If the **window** is NULL it'll be created.

If **width** or **height** is 0, it is set to the size specified inside the avi file being played.

If **callback** is NULL the frames won't be processed, just displayed as is.
You can also define your callback function void callback(IplImage* **img**) and pass a pointer to it as a **callback** parameter. It will be called before displaying each frame with the **img** parameter containing the data of frame going to be displayed. IplImage structure is declared in "ipl.h", which is included from "cv.h" as follows:

```
typedef struct _IplImage {
    int         nSize;          /* size of iplImage struct      */
    int         ID;             /* version                      */
    int         nChannels;
    int         alphaChannel;
    int         depth;          /* pixel depth in bits          */
    char        colorModel[4];
    char        channelSeq[4];
    int         dataOrder;
    int         origin;
    int         align;          /* 4 or 8 byte align            */
    int         width;
    int         height;
```

```
    struct _IplROI *roi;
    struct _IplImage
                *maskROI;          /* poiner to maskROI if any       */
    void        *imageId;          /* use of the application         */
    struct
      _IplTileInfo *tileInfo;        /* contains information on tiling  */
    int         imageSize;         /* useful size in bytes           */
    char        *imageData;        /* pointer to aligned image       */
    int         widthStep;         /* size of aligned line in bytes   */
    int         BorderMode[4];     /*                                 */
    int         BorderConst[4];    /*                                 */
    char        *imageDataOrigin;  /* ptr to full, nonaligned image   */
} IplImage;
```

This function blocks the calling thread until the file finishes or error occurs and then returns.

cvcamResume

Declaration:

int cvcamResume();

Parameters:

None.

Return Value:

Always 0.

Remarks:

Resumes the video if it is paused.

cvcamSelectCamera

Declaration:

int cvcamSelectCamera(int** **out**);

Parameters:

int** **out** (out) – an address of a pointer to int or NULL
If not NULL will contain an array of selected cameras indexes.

Return Value:

The number of selected cameras – 0, 1 or 2.

Remarks:

This function pops up a dialog box, which suggests to select one or 2 cameras to work with. If **out** is not NULL, and user has selected some camera(s), the array of their indexes is placed into ***out**. Use free() to release this array after you don't need it more.

cvcamSetProperty

Declaration:

```
int cvcamSetProperty(int camera, const char* property, void* value);
```

Parameters:

int **camera** (in) – a number of the camera in 0-based index of cameras found in the system.

const char* **property** (in) – a name of the property (See cvcam Properties Interface for details).

void* **value** (in) – depends on the property's name. (See cvcam Properties Interface for details).

Return Value:

0 on success, negative error code for error.

Remarks:

Sets the value of the specified **property** of the specified **camera** to **value**. (See cvcam Properties Interface for details).

cvcamStart

Declaration:

```
int cvcamStart();
```

Parameters:

None.

Return Value:

0 on success, -1 on failure.

Remarks:

Starts the video stream for all enabled cameras.

cvcamStop

Declaration:

```
int cvcamStop();
```

Parameters:

None.

Return Value:

Always 0.

Remarks:

Stops the video stream.

cvcam Properties Interface

There is a list of properties which can be set for each camera. There are 2 functions to work with them:

```
cvcamGetProperty(int cameraindex, const char* property, void* value);
cvcamSetProperty(int cameraindex, const char* property, void* value);
```

The first argument is the number that identifies the camera, the second is the name of the property. The actual type of the 3-rd argument depends on the property. Some of the properties can be set and got, some only set or only got. See the table below for detailes

Property name	Action	argument	Can be get/set	Platform Win/Lin
CVCAM_PROP_ENABLE	Selects/deselects the camera	CVCAMTRUE/ CVCAMFALSE For Set, pointer to int for Get	SG	WL
CVCAM_PROP_RENDER	Renders the cam	CVCAMTRUE/ CVCAMFALSE For Set, pointer to int for Get	SG	WL
CVCAM_PROP_WINDOW	Selects a window for rendering camera's stream	Pointer to HWND(Win) or pointer to Window(Lin)	SG	WL
CVCAM_PROP_CALLBACK	Sets a callback function which processes every frame	void (*callback)(IplImage* image)	S	WL
CVCAM_DESCRIPTION	Gets string name and some other info about the camera	A pointer to struct CameraDescription Under windows only DeviceDescription field is active	G	WL
CVCAM_VIDEOFORMAT	Displays videoformat dialog box for the camera	NULL	G	WL
CVCAM_CAMERAPROPS	Displays camera properties dialog box for the camera	NULL	G	WL
CVCAM_RNDWIDTH	Sets the width of the output video	Pointer to int containing width	SG	WL
CVCAM_RNDHEIGHT	Sets the height of the output video	Pointer to int containing height	SG	WL
CVCAM_STEREO_CALLBACK	Sets the callback	Void	S	W

	which will be called on every two synchronous frames from 2 cameras	(*callback)(IplImage* Image1, IplImage* image2)		
CVCAM_PROP_RAW	Gets the last frame	IplImage**	G	L
CVCAM_PROP_SETFORMAT	Sets video format in non-dialog way	A pointer to struct videoformat.For Set should be initialized with desired values	SG	L