Frequently Asked Questions

How to set image data to OpenCV image buffer

#5400-0384　Version 1.0.0.1

TOSHIBA TELI CORPORATION

Document in Japanese

このドキュメントは TeliCamSDK を使用して取得した画像データを OpenCV の画像バッファに設定する方法について記述したドキュメントです。日本語ドキュメントは英語ドキュメントの後にあります。

End of document in Japanese

This document describes how to set image data acquired using TeliCamSDK to image buffer of OpenCV library..

1. Introduction

OpenCV is a computer vision library on open source basis.

OpenCV provides “IplImage” structure for handling image data in C language, and “cv::Mat” class for handling image data in C++ language

OpenCV has “videoio” module for capturing images from cameras. But “videoio” module does support USB3 Vision and GigE Vision camera.

This document shows how to set image data, captured from USB3 Vision or GigE Vision camera using API of TeliCamSDK on Microsoft Windows, to “IplImage” structure or “cv::Mat” class object.

2. Required software

This document assumes that you use Visual studio 2010 as software development tool. The following libraries should be installed in the PC that Visual studio 2010 is installed.

Library name	Remarks
TeliCamSDK PkgVer 2.1.1.1 or later	TeliCamSDK is downloadable from the following URL. https://www.toshiba-teli.co.jp/cgi/ss/en/service.cgi
OpenCV	Explanation of this document assumes that OpenCV version 2.4.9 is installed in the folder directly under the C drive. Building binary of OpenCV library might be required depending on combination of OpenCV version and Visual Studio version. Please refer to http://opencv.org/ about OpenCV

3. Constructing software using OpenCV library

This chapter introduces the procedure to construct software for displaying the image, acquired from a camera, in OpenCV image display window, based on GrabStreamSimple sample code attached to TeliCamSDK.

The original GrabStreamSimple is a console application which sends software trigger command to a camera and captures an image sent from the camera.

The modified GrabStreamSimple will capture images from a camera in free-run mode, convert captured image from the camera to format of image data used in OpenCV, and show the converted image in OpenCV image window.

In this document, code part of GrabStreamSimple.cpp is shown enclosed in a ruled line. The code stated in a pale color such as gray is a code unnecessary to change. The code written in bold in dark color will be the code part added or modified.

3.1. Copying GrabStreamSimple sample code

GrabStreamSimple sample code is arranged in following directory as Visual Studio 2005 solution.
[Directory that TeliCamSDK is installed]/Samples/VS2005/CPP/GrabStreamSimple

- Copy GrabStreamSimple sample code to another folder.
  Copy to folder that login user has right to read and write files.

- Open GrabStreamSimple.sin in the copied GrabStreamSimple folder using Visual Studio.
  Visual Studio conversion wizard dialog will appear.

- Click [finish] button in the dialog.
  Visual Studio 2005 format solution files will be converted to Visual Studio 2010 format solution files.

- Select “Debug”→”Start Without Debugging” to make sure that conversion succeeded.
  Connect a camera to the PC and select "Start Without Debugging". Command prompt window will appear after compiling the converted solution. Confirm that GrabStreamSimple process works correctly.

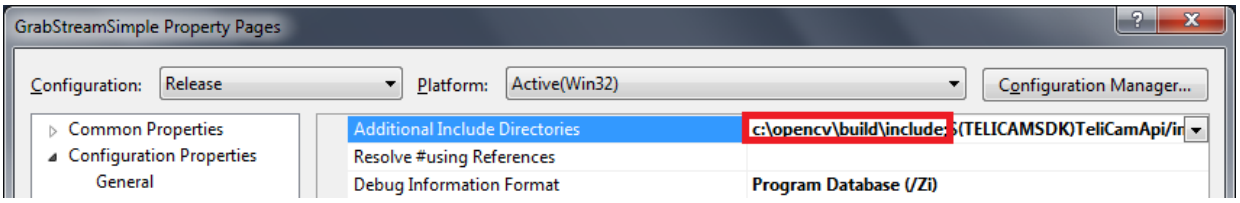
3.2. Making OpenCV library available in GrabStreamSimple project

Modify property of GrabStreamSimple project to make header files and Lib files of OpenCV library readable from GrabStreamSimple project.

- Right click GrabStreamSimple project in Solution Explorer to show property window of it.

- Add the directory which contains include file of OpenCV library to “Additional include Directories” under “Configuration Properties”→”C/C++”→”General”

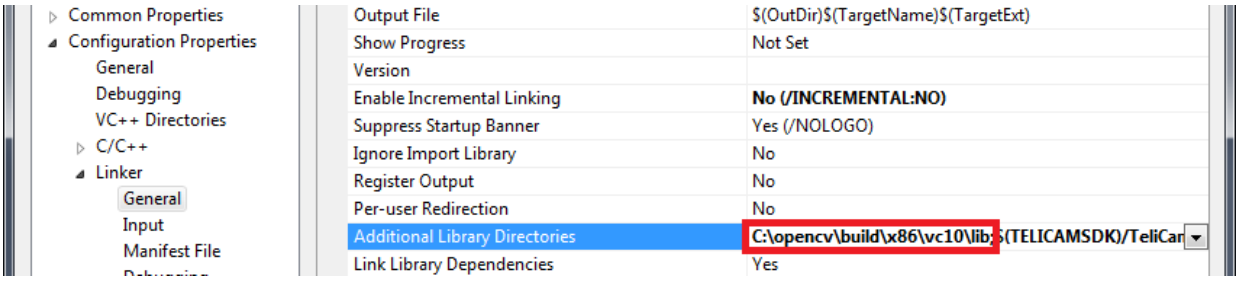
  Add “c:/opencv/build/include;” when OpenCV files are installed in “c:\opencv\”.



- Add the directory which contains library files of OpenCV to “Additional Library Directories” under “Configuration Properties”→”Linker”→”General”.

  Lib files compiled for each version of Visual Studio exist in the folder that OpenCV version 2.4.9 is installed. Add directory that corresponds to the Visual Studio you use to "Additional Library Directories".

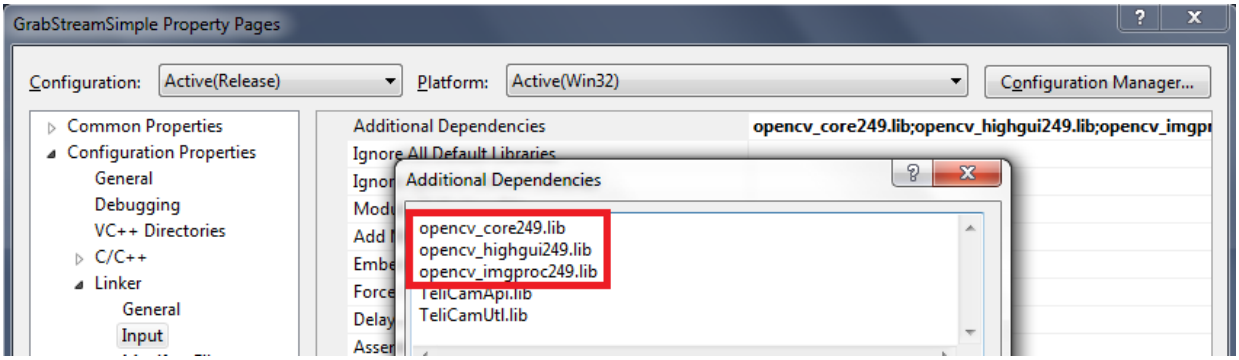
	32bit (x86)	64bit (x64)
Visual Studio 2010	c:/opencv/build/x86/vc10/lib	c:/opencv/build/x64/vc10/lib
Visual Studio 2012	c:/opencv/build/x86/vc11/lib	c:/opencv/build/x64/vc11/lib
Visual Studio 2013	c:/opencv/build/x86/vc12/lib	c:/opencv/build/x64/vc12/lib



- Add library files to “Additional Dependencies” under “Configuration Properties”→”Linker”→”Input”.

  Add the following libraries that are necessary in handling image data in OpenCV, in this document.

  opencv_core249.lib (opencv_core249d.lib in debug configuration)
  opencv_highgui249.lib (opencv_highgui249d.lib in debug configuration)
  opencv_imgproc249.lib (opencv_imgproc249d.lib in debug configuration)



3.3. Adding #Include directive for OpenCV in GrabStreamSimple.cpp

- Open GrabStreamSimple.cpp and add #include directive for opencv.hpp.

  Open GrabStreamSimple.cpp and add directive for including opencv2/opencv, hpp before “#include "TeliCamApi.h"“row to make classes and functions in OpenCV available.

```
#include "opencv2/opencv.hpp"
```

3.4. Adding declaration of OpenCV image data object

- Add declaration of OpenCV image data object in GrabStreamSimple.cpp.

  In GrabStreamSimple, you should decide which one of IplImage structure or cv::Mat class object to use. When you use IplImage structure, delete "//" from the first line "// # define USE_IPL_IMAGE" to activate the #define preprocessor directive.

```
//#define USE_IPL_IMAGE

#ifdef USE_IPL_IMAGE
static ::IplImage        *s_pIplImage;
#else
static cv::Mat           s_openCvImage;
#endif
```

3.5. Changing trigger mode to Free-Run mode

The original GrabStreamSimple sample code uses software trigger mode in capturing images. Modify GrabStreamSimple.cpp to used Free-Run mode in capturing images. Name of `SetSoftTriggerMode()` function is changed to “`SetTriggerMode`” in this document.

- Modify `SetSoftTriggerMode()` function to set Free-Run mode

  The original `SetSoftTriggerMode()` function contains code for setting software trigger mode.

  Replace the second argument of `SetCamTriggerMode()` function with false.

```
s_uiStatus = SetCamTriggerMode(s_hCam, false);
```

- Comment out (delete) codes for sending software trigger command in ImageHandling().

  Since we use free run mode, sending software trigger command is not necessary.

```
    // Send Software Trigger command.
    //s_uiStatus = ExecuteCamSoftwareTrigger(s_hCam);
    //if (s_uiStatus != CAM_API_STS_SUCCESS)
    //{
    //    return 181;
    //}
```

3.6. Creating / releasing OpenCV image object and image window

- Add code for allocating memory to OpenCV image object in OpenStream()

  Code for allocating memory to image buffer used in the application exist in OpenStream () function. Add code for allocating memory to OpenCV image object here.

  In actual applications, number of channels is set according to the image data format used for image processing. In this section, OpenCV image uses the BGR format, and number of channels is set to 3.

- Add code for creating OpenCV image window

  Add code for creating OpenCV image window at the end of `OpenStream()` function.

```
uint32_t OpenStream()
{
    printf("\n");
    printf("OpenStream() started!\n");

    // Create completion event for stream.
    s_hStrmEvt = CreateEvent(NULL, FALSE, FALSE, NULL);
    if (s_hStrmEvt == NULL)
    {
        return 80;
    }

    // Open stream channel.
    // Use default MaxPacketSize(U3v : 65536 or 524288 , GEV : 1500).
    // Payload size will be written to uiPyld,
    s_uiStatus = Strm_OpenSimple(s_hCam, &s_hStrm, &s_uiImgBufSize, s_hStrmEvt);
    if (s_uiStatus != CAM_API_STS_SUCCESS)
    {
        return 81;
    }
    printf(" Payload size : %d[byte]\n", s_uiImgBufSize);

    // Allocate image bufer for receiving image data from TeliCamAPI.
    s_pucImgBuf = (uint8_t *)VirtualAlloc(NULL,
                                s_uiImgBufSize,
                                MEM_RESERVE | MEM_COMMIT,
                                PAGE_EXECUTE_READWRITE);
    if (s_pucImgBuf == NULL)
    {
        return 82;
    }

    // Allocate memory to OpenCV image object.
    //   Gets width and height of image.
    uint32_t uiWidth, uiHeight;

    s_uiStatus = GetCamWidth(s_hCam, &uiWidth);
    if (s_uiStatus != CAM_API_STS_SUCCESS)
    {
        return 900;
    }

    s_uiStatus = GetCamHeight(s_hCam, &uiHeight);
    if (s_uiStatus != CAM_API_STS_SUCCESS)
    {
        return 901;
    }

    //    Allocate memory to OpenCV image object.
#ifdef USE_IPL_IMAGE
    s_pIplImage = ::cvCreateImage( ::cvSize(uiWidth,uiHeight), IPL_DEPTH_8U, 3);
#else
    s_openCvImage = cv::Mat(cv::Size(uiWidth, uiHeight), CV_8UC3, cv::Scalar::all(0));
#endif


    // Create OpenCV image window for showing image.
    cv::namedWindow("OpenCV Test", CV_WINDOW_AUTOSIZE);

    printf("OpenStream() successful.\n");
    return 0;
}
```

- Add code for releasing memory allocated to OpenCV image object (`CloseStream()`)

  Add code for releasing memory allocated to OpenCV image object.

- Add code for disposing OpenCV image window(`CloseStream()`)

  Add code for disposing OpenCV image window.

```
uint32_t CloseStream()
{
    printf("\n");
    printf("CloseStream() started!\n");

    // Close stream.
    if (s_hStrm != NULL)
    {
        Strm_Close(s_hStrm);
        s_hStrm = NULL;
    }

    // Close completion event for stream.
    if (s_hStrmEvt != NULL)
    {
        CloseHandle(s_hStrmEvt);
        s_hStrmEvt = NULL;
    }

    // Release image buffer.
    if (s_pucImgBuf != NULL)
    {
        VirtualFree(s_pucImgBuf, 0, MEM_RELEASE);
        s_pucImgBuf = NULL;
    }

    // Release image buffer of OpenCV image object.
#ifdef USE_IPL_IMAGE
    if (s_pIplImage != NULL)
    {
        ::cvReleaseImage( &s_pIplImage);
        s_pIplImage = NULL;
    }
#else
    // Do Nothing
#endif

    // Destroy OpenCV image window.
    cv::destroyWindow("OpenCV Test");

    printf("CloseStream() successful.\n");
    return 0;
}
```

3.7. Modifying code for handling received images

The code for handling images sent from the camera is contained in `ImageHandling ()` function.

ImageHandling() function in the original GrabStreamSimple transmits software trigger command once and displays the value of the first 8 bytes of the received 1 image.

The modified ImageHandling() function, will execute the process to convert image data sent from the camera sequentially in free-run mode to OpenCV image object and display in the OpenCV image display window, until the escape key is pressed.

- Add **`while`** statement in image receiving process.
  Add "**`while`**" statement to enable receiving image sequentially in free run mode.

- Add code for copying received image data to OpenCV image object.
  Camera utility function `ConvImage()` is used for converting image data received from the camera to BGR format and setting converted image data to OpenCV image object.
  Change the fourth argument of `Strm_ReadCurrentImage()` from NULL to "&imageInfo" to get image width and height. Image width and height are used in `ConvImage()` function.

- Add code for showing image of OpenCV image object in OpenCV image window.
  Use `::cvShowImage()` or `cv::imshow()` for showing image data in OpenCV image window. Specified image data will be drawn when `waitkey()` function is executed.

- Add code for breaking "while" loop.
  Add code for breaking "while" loop when Escape key is pressed.

- Comment out code for printing information

```
uint32_t ImageHandling()
{
    const uint32_t  uiTimeout = 5000;    // 5sec.
    uint32_t        uiRes;
    uint32_t        uiSize  = s_uiImgBufSize;
    // Declare image information variable for getting width and height of image.
    CAM_IMAGE_INFO  imageInfo;
    // Declare key value variable for getting pressed key.
    int             key;

    // Comment out code for sending software trigger command.
    // Send Software Trigger command.
    //s_uiStatus = ExecuteCamSoftwareTrigger(s_hCam);
    //if (s_uiStatus != CAM_API_STS_SUCCESS)
    //{
    //    return 181;
    //}

    // Add while loop for receiving images continuously.
    while(1)
    {
       // Wait for receiving image completion event.
       uiRes = WaitForSingleObject(s_hStrmEvt, uiTimeout);
       if (uiRes != WAIT_OBJECT_0)
       {    // Timeout.
           return 182;
       }

        // Replace the fourth argument with "&imageInfo" to get image size.
       // Copy received image data to local buffer, and get size of received image.
       s_uiStatus = Strm_ReadCurrentImage(s_hStrm, s_pucImgBuf, &uiSize, &imageInfo);
```

```
        if (s_uiStatus != CAM_API_STS_SUCCESS)
        {
            return 183;
        }
        // Comment out code for printing information.
        //printf("Received ImageAcquired event.\n");

#ifdef USE_IPL_IMAGE
        // Copy converted image data to OpenCV image object.
        s_uiStatus = ConvImage(DST_FMT_BGR24, imageInfo.uiPixelFormat, true,
                               s_pIplImage->imageData, s_pucImgBuf,
                               imageInfo.uiSizeX, imageInfo.uiSizeY);
        if (s_uiStatus != CAM_API_STS_SUCCESS)
        {
            return 910;
        }

        // Show image in OpenCV window.
        ::cvShowImage("OpenCV Test", s_pIplImage);
#else
        // Copy converted image data to OpenCV image object.
        s_uiStatus = ConvImage(DST_FMT_BGR24, imageInfo.uiPixelFormat, true,
                               s_openCvImage.data, s_pucImgBuf,
                               imageInfo.uiSizeX, imageInfo.uiSizeY);
        if (s_uiStatus != CAM_API_STS_SUCCESS)
        {
            return 910;
        }

        // Show image in OpenCV window.
        cv::imshow("OpenCV Test", s_openCvImage);
#endif

        // Add code for checking that Escape key is pressed.
        key = cv::waitKey(10);
        if (key == 0x1b)    // [ESC] key
        {
            break;
        }
    }

    // Comment out code for printing leading 8 byte data of the received image data.
    // Display the first 8 pixel values.
    //printf("ImageBuffer :");
    //for (int i = 0; i < 8; i++)
    //{
    //    printf(" %02X ", s_pucImgBuf[i]);
    //}
    //printf("...\n\n");

    return 0;
}
```

3.8. Deleting ProcessLoop() function

The original `ProcessLoop ()` function contains the code to execute the process specified according to key input.

In order to execute only the process of receiving and displaying the image from the camera in free-run mode, replace the code calling `ProcessLoop ()` in `_tmain()` function with the code calling `ImageHandling ()`, and delete `ProcessLoop()` function.

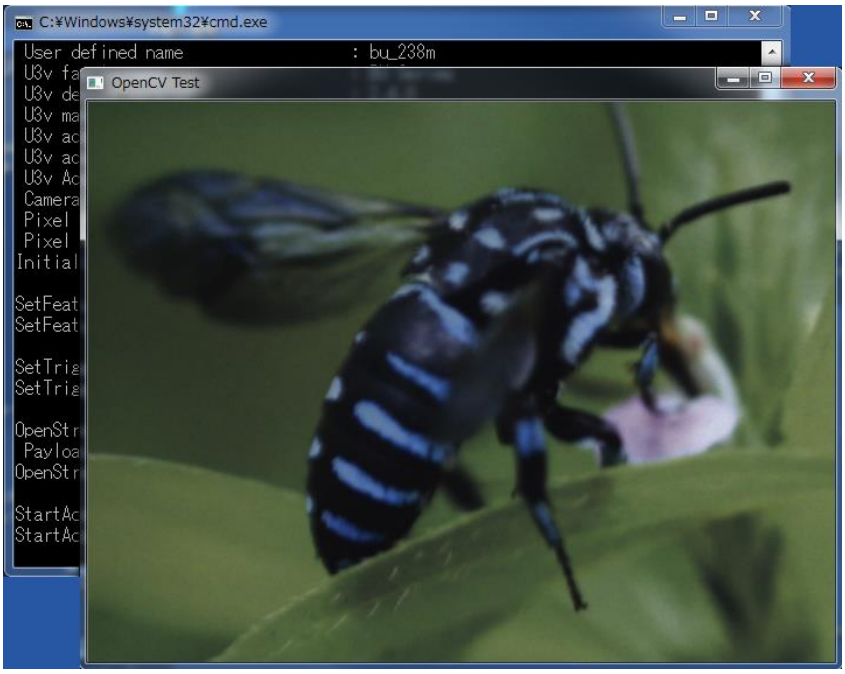
```
        // Start acquisition
       uiStatus = StartAcquisition();
       if (uiStatus != 0)
       {
           printf("Failed StartAcquisition(), Location = %d, Status = 0x%08X.\n", uiStatus,
s_uiStatus);
           break;
       }

       // Receive images.
       uiStatus = ImageHandling();
       if (uiStatus != 0)
       {
           printf("Failed ImageHandling(), Location = %d, Status = 0x%08X.\n", uiStatus,
s_uiStatus);
       }

       // Stop acquisition.
       uiStatus = StopAcquisition();
       if(uiStatus != 0)
       {
           printf("Failed StopAcquisition(), Location = %d, Status = 0x%08X.\n", uiStatus,
s_uiStatus);
           break;
       }
```

3.9. Checking that GrabStreamSimple is modified correctly

After finished editing GrabStreamSimple project, select “Debug”→”Start Without Debugging” to check that images received from the camera are displayed in OpenCV image window.



If error message of “no OpenCV DLL found”is displayed, path of OpenCV DLL file might not be set in the environment variable PATH.

Please add folder that contains path OpenCV DLLs for Visual Studio of the version you use to the environment variable PATH.

	32bit (x86)	64bit (x64)
Visual Studio 2010	c:/opencv/build/x86/vc10/bin	c:/opencv/build/x64/vc10/bin
Visual Studio 2012	c:/opencv/build/x86/vc11/bin	c:/opencv/build/x64/vc11/bin
Visual Studio 2013	c:/opencv/build/x86/vc12/bin	c:/opencv/build/x64/vc12/bin

The followings show procedure for editing the environment variable PATH.

- ✓ Run “Control Panel” →”System”→”Advanced system settings” to open “System Properties” window.
- ✓ Click [Environment Variables] button to show “Environment Variables” window.
- ✓ Scroll down “System variables” table, select variable “Path” and click [Edit] button to open “Edit System variable” window.
- ✓ Add folder name to value shown in “Variable value:” with semicolon as a separator.
- ✓ Close “Edit System variable” window and other opened windows by clicking [OK] button.

This example code specifies PixelFormat member variable of image information structure received from the camera to the second argument of `ConvImage()` function. Depending on camera model and camera firmware version, ConvImage() function might return error or image might be converted inadequately. In such a case, specify PixelFormat value gotten by GetCamPixelFormat() function instead of PixelFormat member variable of image information structure.

3.10. GrabStreamSimple.cpp after modification

This section shows the modified GrabStreamSimple.cpp. Commented out codes are not shown in this section.

The following modifications are applied in addition to the modification described in the section above.

- Specify 1 to channel count of OpenCV image object when PixelFormat sent from the camera is Mono8.
- When PixelFormat sent from the camera is Mono8, copies received image data to OpenCV image object with ReadCurrentImage() function.

```
#include <windows.h>
#include <stdio.h>
#include <tchar.h>

#include "opencv2/opencv.hpp"

#include "TeliCamApi.h"
#include "TeliCamUtl.h"

using namespace Teli;

/*****************************************************************************/
/* Prototype declares                                                        */
/*****************************************************************************/
uint32_t Initialize();
void     Terminate();
uint32_t SetTriggerMode();
uint32_t OpenStream();
uint32_t CloseStream();
uint32_t StartAcquisition();
uint32_t StopAcquisition();
uint32_t ImageHandling();

/*****************************************************************************/
/* Static Values                                                             */
/*****************************************************************************/
static CAM_API_STATUS  s_uiStatus = CAM_API_STS_SUCCESS;

static uint8_t  *s_pucImgBuf    = NULL;  // Local image buffer for receiving image stream.
static uint32_t s_uiImgBufSize  = 0;     // Size of image buffer.
// Handles
static CAM_HANDLE       s_hCam      = NULL;       // Camera handle
static CAM_STRM_HANDLE  s_hStrm     = NULL;       // Stream handle
static HANDLE           s_hStrmEvt  = NULL;       // Completion event for stream.

static CAM_PIXEL_FORMAT s_pixelFormat;

//#define USE_IPL_IMAGE

#ifdef USE_IPL_IMAGE
static ::IplImage       *s_pIplImage = NULL;
#else
static cv::Mat          s_openCvImage;
#endif

static char             s_szWindowTitle[] = "OpenCV Test";

/*****************************************************************************/
/* Functions                                                                 */
/*****************************************************************************/
int _tmain(int argc, _TCHAR* argv[])
```

```
{
    uint32_t  uiStatus = 0;

    do
    {
        uiStatus = Initialize();
        if (uiStatus != 0)
        {
            printf("Failed Initialize(%d), Status=0x%08X.\n", uiStatus, s_uiStatus);
            break;
        }

        // Set software one-shot trigger mode.
        uiStatus = SetTriggerMode();
        if (uiStatus != 0)
        {
            printf("Failed SetTriggerMode(%d), Status=0x%08X.\n", uiStatus, s_uiStatus);
            break;
        }

        // Open stream.
        uiStatus = OpenStream();
        if (uiStatus != 0)
        {
            printf("Failed OpenStream(%d), Status=0x%08X.\n", uiStatus, s_uiStatus);
            break;
        }

        // Start acquisition
        uiStatus = StartAcquisition();
        if (uiStatus != 0)
        {
            printf("Failed StartAcquisition(%d), Status=0x%08X.\n", uiStatus, s_uiStatus);
            break;
        }

        // Receive images.
        uiStatus = ImageHandling();
        if (uiStatus != 0)
        {
            printf("Failed ImageHandling(%d), Status=0x%08X.\n", uiStatus, s_uiStatus);
        }

        // Stop acquisition.
        uiStatus = StopAcquisition();
        if(uiStatus != 0)
        {
            printf("Failed StopAcquisition(%d), Status=0x%08X.\n", uiStatus, s_uiStatus);
            break;
        }
    } while(0);

    // Close stream.
    CloseStream();

    // Terminate.
    Terminate();

    return 0;
}

uint32_t Initialize()
{
    uint32_t  uiNum;

    // API initialization.
```

```
    s_uiStatus = Sys_Initialize();
    if (s_uiStatus != CAM_API_STS_SUCCESS)
        return 1;

    // Get uiNumber of camera.
    s_uiStatus = Sys_GetNumOfCameras(&uiNum);
    if (s_uiStatus != CAM_API_STS_SUCCESS)
        return 3;

    if (uiNum == 0)
    {
         printf(" No cameras found.\n");
        return 4;
    }
    printf(" %d camera(s) found.\n", uiNum);

    // Open camera that is detected first, in this sample code.
    uint32_t iCamNo = 0;
    s_uiStatus = Cam_Open(iCamNo, &s_hCam);

    if (s_uiStatus != CAM_API_STS_SUCCESS)
        return 6;

    return 0;
}

void Terminate()
{
    // Close camera.
    if (s_hCam != NULL)
        Cam_Close(s_hCam);

    // Terminate system.
    Sys_Terminate();
}

uint32_t SetTriggerMode()
{
    // Set TriggerMode false, in this sample code.
    s_uiStatus = SetCamTriggerMode(s_hCam, false);
    if (s_uiStatus != CAM_API_STS_SUCCESS)
        return 40;

    return 0;
}

uint32_t OpenStream()
{
    // Create completion event for stream.
    s_hStrmEvt = CreateEvent(NULL, FALSE, FALSE, NULL);
    if (s_hStrmEvt == NULL)
        return 80;

    // Open stream channel.
     s_uiStatus = Strm_OpenSimple(s_hCam, &s_hStrm, &s_uiImgBufSize, s_hStrmEvt);
    if (s_uiStatus != CAM_API_STS_SUCCESS)
        return 81;

    uint32_t uiWidth, uiHeight;

    s_uiStatus = GetCamWidth(s_hCam, &uiWidth);
    if (s_uiStatus != CAM_API_STS_SUCCESS)
        return 900;

     s_uiStatus = GetCamHeight(s_hCam, &uiHeight);
    if (s_uiStatus != CAM_API_STS_SUCCESS)
```

```
        return 901;

    s_uiStatus = GetCamPixelFormat(s_hCam, &s_pixelFormat);
    if (s_uiStatus != CAM_API_STS_SUCCESS)
        return 902;

    if (s_pixelFormat == PXL_FMT_Mono8)
    {
#ifdef USE_IPL_IMAGE
        s_pIplImage = ::cvCreateImage( ::cvSize(uiWidth,uiHeight), IPL_DEPTH_8U, 1);
#else
        s_openCvImage = cv::Mat(cv::Size(uiWidth, uiHeight), CV_8UC1, cv::Scalar::all(0));
#endif
    } else
    {
        // Allocate image bufer for receiving image data from TeliCamAPI.
        s_pucImgBuf = (uint8_t *)VirtualAlloc(NULL,
                                        s_uiImgBufSize,
                                        MEM_RESERVE | MEM_COMMIT,
                                        PAGE_EXECUTE_READWRITE);
        if (s_pucImgBuf == NULL)
            return 82;

#ifdef USE_IPL_IMAGE
        s_pIplImage = ::cvCreateImage( ::cvSize(uiWidth,uiHeight), IPL_DEPTH_8U, 3);
#else
        s_openCvImage = cv::Mat(cv::Size(uiWidth, uiHeight), CV_8UC3, cv::Scalar::all(0));
#endif
    }

    cv::namedWindow(s_szWindowTitle, CV_WINDOW_AUTOSIZE);

    return 0;
}

uint32_t CloseStream()
{
    // Close stream.
    if (s_hStrm != NULL)
        Strm_Close(s_hStrm);

    // Close completion event for stream.
    if (s_hStrmEvt != NULL)
        CloseHandle(s_hStrmEvt);

    // Release image buffer.
    if (s_pucImgBuf != NULL)
        VirtualFree(s_pucImgBuf, 0, MEM_RELEASE);

#ifdef USE_IPL_IMAGE
    if (s_pIplImage != NULL)
        ::cvReleaseImage( &s_pIplImage);
#else
    // Do Nothing
#endif

    cv::destroyWindow(s_szWindowTitle);

    return 0;
}

uint32_t StartAcquisition()
{
    s_uiStatus = Strm_Start(s_hStrm);
    if (s_uiStatus != CAM_API_STS_SUCCESS)
        return 100;
```

```
    return 0;
}

uint32_t StopAcquisition()
{
    s_uiStatus = Strm_Stop(s_hStrm);
    if (s_uiStatus != CAM_API_STS_SUCCESS)
        return 120;

    return 0;
}

uint32_t ImageHandling()
{
    const uint32_t  uiTimeout = 5000;    // 5sec.
    uint32_t        uiRes;
    uint32_t        uiSize;
    CAM_IMAGE_INFO  imageInfo;
    int             key;

    while(1)
    {
        // Wait for receiving image completion event.
        uiRes = WaitForSingleObject(s_hStrmEvt, uiTimeout);
        if (uiRes != WAIT_OBJECT_0)
            return 182;     // Timeout.

        if (s_pixelFormat == PXL_FMT_Mono8)
        {
#ifdef USE_IPL_IMAGE
            s_uiStatus = Strm_ReadCurrentImage(s_hStrm, s_pIplImage->imageData, &uiSize,
&imageInfo);
            if (s_uiStatus != CAM_API_STS_SUCCESS)
                return 183;

            ::cvShowImage(s_szWindowTitle, s_pIplImage);
#else
            s_uiStatus  =  Strm_ReadCurrentImage(s_hStrm,  s_openCvImage.data,  &uiSize,
&imageInfo);
            if (s_uiStatus != CAM_API_STS_SUCCESS)
                return 183;

            cv::imshow(s_szWindowTitle, s_openCvImage);
#endif
        } else
        {
            // Copy received image data to local buffer, and get size of received image.
            s_uiStatus = Strm_ReadCurrentImage(s_hStrm, s_pucImgBuf, &uiSize, &imageInfo);
            if (s_uiStatus != CAM_API_STS_SUCCESS)
                return 183;

            // Convert image format.
#ifdef USE_IPL_IMAGE
            s_uiStatus = ConvImage(DST_FMT_BGR24, imageInfo.uiPixelFormat, true,
                s_pIplImage->imageData, s_pucImgBuf, imageInfo.uiSizeX, imageInfo.uiSizeY);
            if (s_uiStatus != CAM_API_STS_SUCCESS)
                return 910;

            ::cvShowImage(s_szWindowTitle, s_pIplImage);
#else
            s_uiStatus = ConvImage(DST_FMT_BGR24, imageInfo.uiPixelFormat, true,
                s_openCvImage.data, s_pucImgBuf, imageInfo.uiSizeX, imageInfo.uiSizeY);
            if (s_uiStatus != CAM_API_STS_SUCCESS)
                return 910;
```

```
            cv::imshow(s_szWindowTitle, s_openCvImage);
#endif
        }

        key = cv::waitKey(10);
        if (key == 0x1b)    // [ESC] key
            break;
    }

    return 0;
}
```

1. はじめに

OpenCV は、オープンソースのコンピュータ・ビジョンライブラリです。

OpenCV は C 言語で画像データを扱うための IplImage 構造体と、C++言語で画像データを扱うための cv::Mat クラスを提供しています。

OpenCV にはカメラから画像を取得することができる videoio モジュールを持っていますが、このモジュールは USB3 Vision と GigE Vision の カメラ規格をサポートしておりません。

本ドキュメントは、Microsoft Windows で TeliCamSDK を使用し、USB3 Vision カメラおよび GigE Vision カメラの画像データを OpenCV の IplImage 構造体または cv::Mat クラスオブジェクトとして取り込む方法について説明します。

2. 使用ソフトウェア

本ドキュメントはソフトウェア開発ツールとして Visual Studio 2010 を使用する前提で説明を記載しています。

Visual Studio 2010 がインストールされている PC には以下のライブラリがインストールされている必要があります.

ライブラリ名	備　　考
TeliCamSDK PkgVer 2.1.1.1 or later	TeliCamSDK は以下の URL からダウンロードできます。 https://www.toshiba-teli.co.jp/cgi/ss/jp/service_j.cgi
OpenCV	この文書では OpenCV version 2.4.9 が C ドライブの直下のフォルダにインストールされている前提で説明をしています。 ご使用の OpenCV のバージョンと Visual Studio のバージョンの組み合わせによっては OpenCV ライブラリのビルドが必要になる場合があります。 OpenCV についての情報は http://opencv.org/ をご覧ください。

3. OpenCV ライブラリを使用した画像取得ソフトの作成

この章では、カメラから取得した画像を OpenCV の画像表示ウィンドウに表示するソフトウェアを、TeliCamSDK をインストールするとインストールされる GrabStreamSimple サンプルコードをもとにして作成する手順を紹介します。

GrabStreamSimple サンプルコードは、カメラにソフトウェアトリガコマンドを送り、カメラから送られる画像を取得するコンソールアプリケーションです。

このアプリケーションを、フリーラン動作でカメラから画像を取得し、取得した画像データを OpenCV の IplImage 構造体または cv::Mat クラスオブジェクトのフォーマットに変換し、OpenCV の画像表示ウィンドウに表示するよう改造します。

このドキュメントでは 3.3 節以降に GrabStreamSimple.cpp のコードを抜粋し罫線で囲って掲載しています。灰色などの淡い色で記載されたコードは変更不要なコードです。濃い色で太字で記載されたコードが追加または変更するコード部分になります。

3.1. GrabStreamSimple サンプルコードのコピーとオープン

以下のディレクトリに Visual Studio 2005 のソリューションの形で GrabStreamSimple サンプルコードが配置されています。

[TeliCamSDK インストールディレクトリ]\Samples\VS2005\CPP\GrabStreamSimple

- GrabStreamSimple サンプルコードを別のフォルダにコピーしてください。
  読み書き可能なフォルダにコピーしてください。

- Visual Studio で、コピーした GrabStreamSimple フォルダ内の GrabStreamSimple.sln を開いてください。
  Visual Studio 変換ウィザードダイアログが表示されます。

- 変換ウィザードダイアログの[完了] ボタンをクリックしてください。
  Visual Studio 2005 のソリューションが Visual Studio 2010 のソリューションに変換されます。

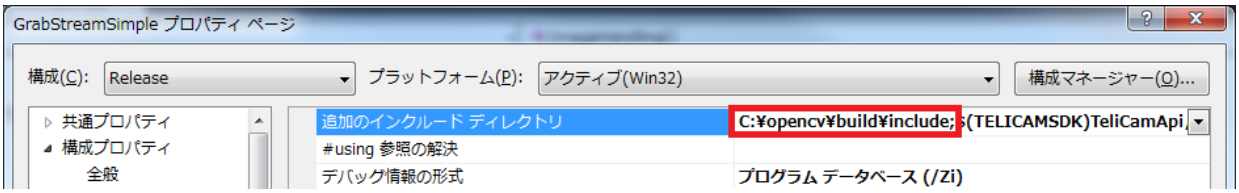
- 「デバッグ」→「デバッグなしで実行」を選択し、正常に変換されたことを確認してください。
  カメラを PC に接続してから「デバッグなしで実行」を選択してください。ソリューションがコンパイルされた後、コンソール画面が開き、GrabStreamSimple の処理が正常に動作すれば、変換成功です。

3.2. GrabStreamSimple プロジェクトで OpenCV をライブラリ利用可能に設定

GrabStreamSimple プロジェクトのプロパティで OpenCV ライブラリのヘッダファイルと Lib ファイルを読み込めるように設定します。

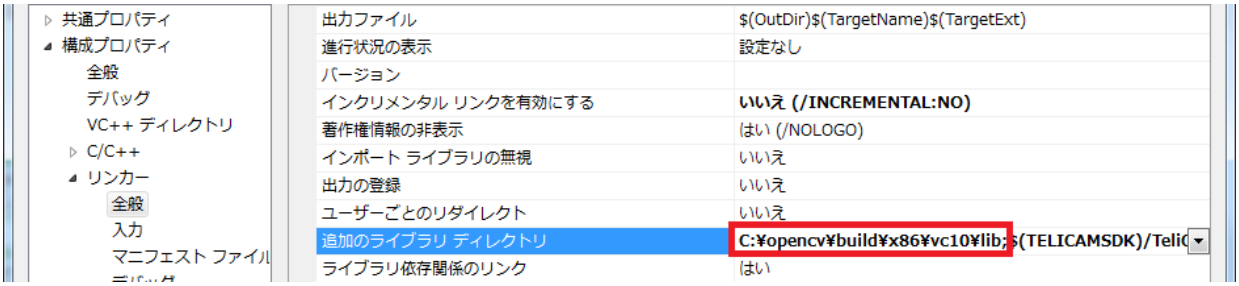
- ソリューションエクスプローラーで GrabStreamSimple プロジェクトを右クリックし、プロジェクトのプロパティ画面を表示させてください。

- 「構成プロパティ」→「C/C++」→「 全般」の「追加のインクルード　ディレクトリ」に OpenCV ライブラリの include ファイルが格納されているディレクトリを追加してください。
  OpenCV が c:\opencv\にインストールされているときは「c:\opencv\build\include;」を追加して下さい。

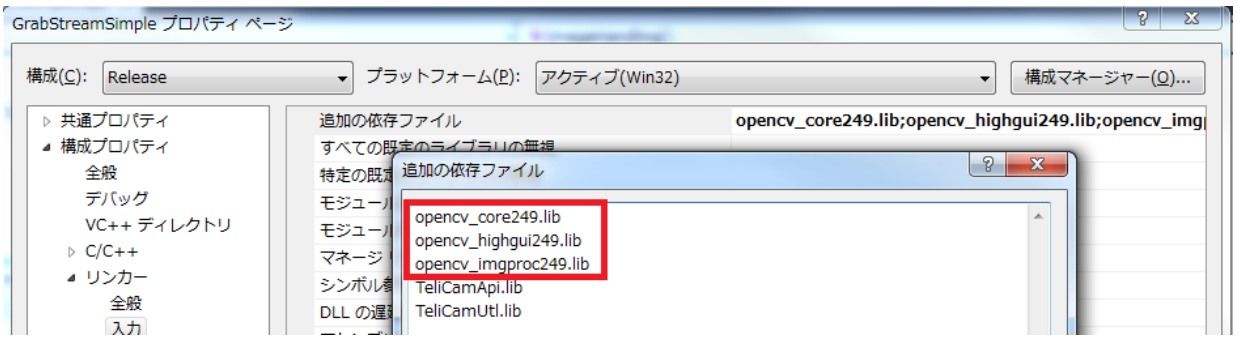


- 「構成プロパティ」→「リンカー」→「 全般」の「追加のライブラリ　ディレクトリ」に、OpenCV のライブラリファイルが格納されているディレクトリを追加してください。
  OpenCV version 2.4.9 では Visual Studio の各バージョンにあわせてコンパイルした lib ファイルが予め提供されています。使用する Visual Studio のバージョンにあったディレクトリを追加してください。

	32bit (x86)	64bit (x64)
Visual Studio 2010	c:\opencv\build\x86\vc10\lib	c:\opencv\build\x64\vc10\lib
Visual Studio 2012	c:\opencv\build\x86\vc11\lib	c:\opencv\build\x64\vc11\lib
Visual Studio 2013	c:\opencv\build\x86\vc12\lib	c:\opencv\build\x64\vc12\lib



- 「構成プロパティ」→「リンカー」→「入力」の「追加の依存ファイル」に、使用するライブラリファイルを追加してください。
  本ドキュメントでは、OpenCV で画像を扱うために必要な以下のライブラリを追加します。
  opencv_core249.lib　　（Debug コンフィギュレーションの場合は opencv_core249d.lib）
  opencv_highgui249.lib　（Debug コンフィギュレーションの場合は opencv_highgui249d.lib）
  opencv_imgproc249.lib （Debug コンフィギュレーションの場合は opencv_imgproc249d.lib）



3.3. Opencv ヘッダファイルの#include ディレクティブの追加

➢ Opencv.hpp への#include ディレクティブの GrabStreamSimple.cpp への追加

GrabStreamSimple.cpp を開き、「`#include "TeliCamApi.h"`」の行の前に opencv2/opencv.hpp を組み込むディレクティブを追加して、OpenCV のクラスと関数を使用できるようにします。

```
#include "opencv2/opencv.hpp"
```

3.4. OpenCV 画像データを格納する変数の宣言追加

本ドキュメントでは `IplImage` 構造体と cv::Mat クラスオブジェクトのどちらか一方を宣言するようにしています。`IplImage` 構造体を使用する場合は、先頭行の「`//#define USE_IPL_IMAGE`」から[「`//`」を削除して`#define` プリプロセッサディレクティブを有効にしてください。

```
//#define USE_IPL_IMAGE

#ifdef USE_IPL_IMAGE
static ::IplImage        *s_pIplImage;
#else
static cv::Mat           s_openCvImage;
#endif
```

3.5. トリガモードをフリーランモードに変更

オリジナルの GrabStreamSimple サンプルコードはソフトウェアトリガモードを使用して画像を取得しています。本ドキュメントではフリーランモードで画像を取得するよう処理を変更します。

本ドキュメントでは `SetSoftwareTriggerMode()`関数は `SetTriggerMode()`に名称変更しています。

➢ `SetSoftwareTriggerMode()`関数をフリーランモードに設定する関数に変更

オリジナルの `SetSoftwareTriggerMode()`はカメラがソフトウェアトリガモードで動作するよう設定しています。`SetCamTriggerMode()`関数の第２引数を `false` に差し替えることによりフリーランモードでカメラが動作するようになります。

```
s_uiStatus = SetCamTriggerMode(s_hCam, false);
```

➢ `ImageHandling()`関数内のソフトウェアトリガコマンドを送信するコードの削除。

フリーランモードを使用するよう変更したので、この処理は不要です。

```
    // Send Software Trigger command.
    //s_uiStatus = ExecuteCamSoftwareTrigger(s_hCam);
    //if (s_uiStatus != CAM_API_STS_SUCCESS)
    //{
    //    return 181;
    //}
```

3.6. OpenCV 画像オブジェクトと画像表示画面の生成／解放

➢ OpenCV 画像バッファにメモリを割付けるコードの追加（`OpenStream()`関数）

`OpenStream()` 関数の中にアプリケーションが使用する画像バッファのメモリを確保するコードがあります。この関数に OpenCV の画像バッファのメモリを確保するコードを追加してください。

実際のアプリケーションでは画像処理で使用する画像データフォーマットに合わせてチャンネル数を設定しますが、ここでは OpenCV の画像に BGR フォーマットを使用することとし、チャンネル数には 3 を設定しています。

➢ OpenCV の画像表示ウィンドウを作成するコードの追加（OpenStream()関数）

OpenStream()関数の最後の部分に OpenCV の画像表示ウィンドウを作成するコードを追加してください。

```
uint32_t OpenStream()
{
    printf("\n");
    printf("OpenStream() started!\n");

    // Create completion event for stream.
    s_hStrmEvt = CreateEvent(NULL, FALSE, FALSE, NULL);
    if (s_hStrmEvt == NULL)
    {
        return 80;
    }

    // Open stream channel.
    // Use default MaxPacketSize(U3v : 65536 or 524288 , GEV : 1500).
    // Payload size will be written to uiPyld,
    s_uiStatus = Strm_OpenSimple(s_hCam, &s_hStrm, &s_uiImgBufSize, s_hStrmEvt);
    if (s_uiStatus != CAM_API_STS_SUCCESS)
    {
        return 81;
    }
    printf(" Payload size : %d[byte]\n", s_uiImgBufSize);

    // Allocate image bufer for receiving image data from TeliCamAPI.
    s_pucImgBuf = (uint8_t *)VirtualAlloc(NULL,
                                s_uiImgBufSize,
                                MEM_RESERVE | MEM_COMMIT,
                                PAGE_EXECUTE_READWRITE);
    if (s_pucImgBuf == NULL)
    {
        return 82;
    }

    // Allocate memory to OpenCV image object.
    //   Gets width and height of image.
    uint32_t uiWidth, uiHeight;

    s_uiStatus = GetCamWidth(s_hCam, &uiWidth);
    if (s_uiStatus != CAM_API_STS_SUCCESS)
    {
        return 900;
    }

    s_uiStatus = GetCamHeight(s_hCam, &uiHeight);
    if (s_uiStatus != CAM_API_STS_SUCCESS)
    {
        return 901;
    }

    //    Allocate memory to OpenCV image object.
#ifdef USE_IPL_IMAGE
    s_pIplImage = ::cvCreateImage( ::cvSize(uiWidth,uiHeight), IPL_DEPTH_8U, 3);
#else
    s_openCvImage = cv::Mat(cv::Size(uiWidth, uiHeight), CV_8UC3, cv::Scalar::all(0));
#endif


    // Create OpenCV window for showing image.
    cv::namedWindow("OpenCV Test", CV_WINDOW_AUTOSIZE);

    printf("OpenStream() successful.\n");
    return 0;
}
```


➢ OpenCV の画像バッファに割り付けたメモリを解放するコードの追加（CloseStream() 関数）

OpenStream()関数内で割付けた OpenCV 画像バッファのメモリを開放します。

➢ OpenCV の画像表示ウィンドウを解放するコードの追加（CloseStream()関数）

OpenStream()関数内で生成した OpenCV 画像表示ウィンドウを破棄します。

```
uint32_t CloseStream()
{
    printf("\n");
    printf("CloseStream() started!\n");

    // Close stream.
    if (s_hStrm != NULL)
    {
        Strm_Close(s_hStrm);
        s_hStrm = NULL;
    }

    // Close completion event for stream.
    if (s_hStrmEvt != NULL)
    {
        CloseHandle(s_hStrmEvt);
        s_hStrmEvt = NULL;
    }

    // Release image buffer.
    if (s_pucImgBuf != NULL)
    {
        VirtualFree(s_pucImgBuf, 0, MEM_RELEASE);
        s_pucImgBuf = NULL;
    }

   // Release image buffer of OpenCV image object.
#ifdef USE_IPL_IMAGE
    if (s_pIplImage != NULL)
    {
        ::cvReleaseImage( &s_pIplImage);
        s_pIplImage = NULL;
    }
#else
    // Do Nothing
#endif

    // Destroy OpenCV image window.
   cv::destroyWindow("OpenCV Test");

    printf("CloseStream() successful.\n");
    return 0;
}
```

3.7. 画像受信処理の変更

カメラから送信された画像をアプリケーションに取り込む処理は ImageHandling()関数に記述されています。

オリジナルの GrabStreamSimple ではソフトウェアトリガ信号を 1 回送信し、受信した 1 枚の画像の先頭 8 バイトの値を表示する処理が記載されています。本ドキュメントでは、エスケープキーが押されるまで、カメラからフリーランモードで送信される画像データを逐次 OpenCV 画像オブジェクトに設定し、OpenCV 画像表示ウィンドウに表示するように処理を変更します。

- 画像取得処理に while 文を追加
  フリーランモードで画像を連続的に取得できるようにします。

- 取り込んだ画像データを OpenCV の画像オブジェクトにコピーするコードの追加。
  このコードではカメラユーティリティ関数 ConvImage()でカメラから受信した画像データを BGR フォーマットに変換して OpenCV の画像オブジェクトに設定しています。
  ConvImage()関数は画像の幅と高さを引数に指定する必要があるので、CAM_IMAGE_INFO 構造体をローカル変数として宣言し Strm_ReadCurrentImage()関数の引数に指定して画像情報を取得するよう変更します。

- OpenCV の画像オブジェクト内の画像を OpenCV 画面に表示するコードの追加
  ::cvShowImage()または cv::imshow()で画像を OpenCV 画面に表示するよう指示します。waitKey()関数がコールされたタイミングで OpenCV ウィンドウの描画処理が実行されます。

- While ループから抜けるためのコードの追加
  エスケープキーが押されたら while 文から抜けるコードを追加します。

- 不要となった Print 文のコメントアウト
  ImageAcquired イベント受信表示および受信画像の先頭 8byte の表示は行わないようにします。

```
uint32_t ImageHandling()
{
    const uint32_t  uiTimeout = 5000;    // 5sec.
    uint32_t        uiRes;
    uint32_t        uiSize  = s_uiImgBufSize;
    // Declare image information variable for getting width and height of image.
    CAM_IMAGE_INFO  imageInfo;
    // Declare key value variable for getting pressed key.
    int             key;

    // Comment out code for sending software trigger command.
    // Send Software Trigger command.
    //s_uiStatus = ExecuteCamSoftwareTrigger(s_hCam);
    //if (s_uiStatus != CAM_API_STS_SUCCESS)
    //{
    //    return 181;
    //}

    // Add while loop for receiving images continuously.
    while(1)
    {
        // Wait for receiving image completion event.
        uiRes = WaitForSingleObject(s_hStrmEvt, uiTimeout);
        if (uiRes != WAIT_OBJECT_0)
        {    // Timeout.
            return 182;
        }

         // Replace the fourth argument with "&imageInfo" to get image size.
        // Copy received image data to local buffer, and get size of received image.
```

```
        s_uiStatus = Strm_ReadCurrentImage(s_hStrm, s_pucImgBuf, &uiSize, &imageInfo);
        if (s_uiStatus != CAM_API_STS_SUCCESS)
        {
            return 183;
        }
        // Comment out code for printing information.
        //printf("Received ImageAcquired event.\n");

#ifdef USE_IPL_IMAGE
        // Copy converted image data to OpenCV image object.
        s_uiStatus = ConvImage(DST_FMT_BGR24, imageInfo.uiPixelFormat, true,
                               s_pIplImage->imageData, s_pucImgBuf,
                               imageInfo.uiSizeX, imageInfo.uiSizeY);
        if (s_uiStatus != CAM_API_STS_SUCCESS)
        {
            return 910;
        }

        // Show image in OpenCV window.
        ::cvShowImage("OpenCV Test", s_pIplImage);
#else
        // Copy converted image data to OpenCV image object.
        s_uiStatus = ConvImage(DST_FMT_BGR24, imageInfo.uiPixelFormat, true,
                               s_openCvImage.data, s_pucImgBuf,
                               imageInfo.uiSizeX, imageInfo.uiSizeY);
        if (s_uiStatus != CAM_API_STS_SUCCESS)
        {
            return 910;
        }

        // Show image in OpenCV window.
        cv::imshow("OpenCV Test", s_openCvImage);
#endif

        // Add code for checking that Escape key is pressed.
        key = cv::waitKey(10);
        if (key == 0x1b)    // [ESC] key
        {
            break;
        }
    }

    // Comment out code for printing leading 8 byte data of the received image data.
    // Display the first 8 pixel values.
    //printf("ImageBuffer :");
    //for (int i = 0; i < 8; i++)
    //{
    //    printf(" %02X ", s_pucImgBuf[i]);
    //}
    //printf("...\n\n");

    return 0;
}
```

3.8. ProcessLoop()処理の削除

オリジナルの ProcessLoop()関数にはキー入力に応じて指定された処理を実行するコードが入っています。フリーランモードでカメラから画像を受信し表示する処理だけを実行するようにするために、_tmain()関数の ProcessLoop() 関数をコールしていたコードを mageHandling() 関数をコールするコードに置き換え、ProcessLoop() 関数を削除します。

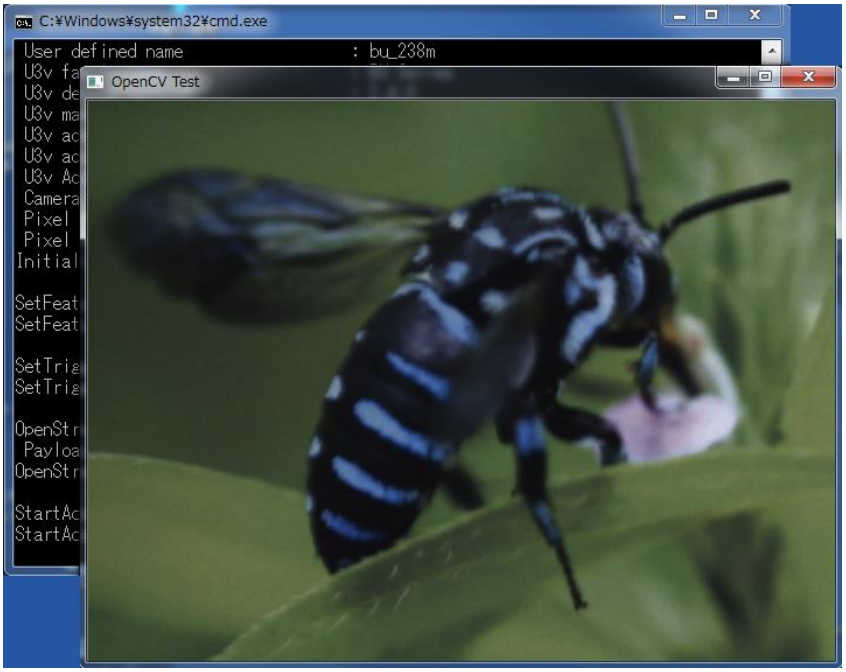
```
        // Start acquisition
       uiStatus = StartAcquisition();
       if (uiStatus != 0)
       {
           printf("Failed StartAcquisition(), Location = %d, Status = 0x%08X.\n", uiStatus,
s_uiStatus);
           break;
       }

       // Receive images.
       uiStatus = ImageHandling();
       if (uiStatus != 0)
       {
           printf("Failed ImageHandling(), Location = %d, Status = 0x%08X.\n", uiStatus,
s_uiStatus);
       }

       // Stop acquisition.
       uiStatus = StopAcquisition();
       if(uiStatus != 0)
       {
           printf("Failed StopAcquisition(), Location = %d, Status = 0x%08X.\n", uiStatus,
s_uiStatus);
           break;
       }
```

3.9. 改造した GrabStreamSimple の動作確認

コードの改造が終わったら、「デバッグ」→「デバッグなしで実行」を選択し、カメラで撮像した画像が OpenCV の画面に表示されることを確認してください。



OpenCV の DLL が見つからないというエラーメッセージが表示された場合は、OpenCV の DLL フォルダのパスが環境変数 PATH に設定されていない可能性があります。

環境変数 PATH に、使用する Visual Studio のバージョンに合った dll が保存されているフォルダを追加してください。

	32bit (x86)	64bit (x64)
Visual Studio 2010	c:\opencv\build\x86\vc10\bin	c:\opencv\build\x64\vc10\bin
Visual Studio 2012	c:\opencv\build\x86\vc11\bin	c:\opencv\build\x64\vc11\bin
Visual Studio 2013	c:\opencv\build\x86\vc12\bin	c:\opencv\build\x64\vc12\bin

環境変数 PATH は以下の手順で編集できます。

- 「コントロールパネル」→「システム」→「システムの詳細設定」でシステムのプロパティを開く。
- [環境変数] ボタンをクリックする。
  環境変数画面が表示されます。
- 「システム環境変数」の表を下にスクロールさせて変数 Path を選択し、[編集]ボタンをクリックする。
- システム変数の編集画面で変数値(V)に表示される値にセミコロン区切りでフォルダ名を追加する。
- ［OK］ボタンをクリックしてシステム変数の編集画面を閉じ、環境変数画面などの今までに開いた画面も［OK］ボタンクリックですべて閉じる。

この例では ConvImage() 関数の第２引数にカメラから受信した画像情報の PixelFormat メンバ変数を指定していますが、カメラの機種とファームウェアバージョンによっては ConvImage() 関数がエラーを返したり、不適切な画像に変換される場合があります。このようなときは画像情報の PixelFormat メンバ変数の代わりに GetCamPixelFormat()関数で取得した PixelFormat 値を指定してください。

3.10. 改造後の GramStreamSimple.cpp

改造後の GramStreamSimple.cpp を以下に示します。
このコードでは前述の変更の他に、カメラから出力される画像フォーマットが Mono8 のときのみ、OpenCV の画像チャンネル数を 1 チャンネルにし、ConvImage()関数を使用しないで Strm_ReadCurrentImage()で取得した画像データを直接 OpenCV の画像メモリにコピーするように変更しています。

```
#include <windows.h>
#include <stdio.h>
#include <tchar.h>

#include "opencv2/opencv.hpp"

#include "TeliCamApi.h"
#include "TeliCamUtl.h"

using namespace Teli;

/*****************************************************************************/
/* Prototype declares                                                        */
/*****************************************************************************/
uint32_t Initialize();
void     Terminate();
uint32_t SetTriggerMode();
uint32_t OpenStream();
uint32_t CloseStream();
uint32_t StartAcquisition();
uint32_t StopAcquisition();
uint32_t ImageHandling();

/*****************************************************************************/
/* Static Values                                                             */
/*****************************************************************************/
static CAM_API_STATUS  s_uiStatus = CAM_API_STS_SUCCESS;

static uint8_t  *s_pucImgBuf    = NULL;  // Local image buffer for receiving image stream.
static uint32_t s_uiImgBufSize  = 0;     // Size of image buffer.
// Handles
static CAM_HANDLE       s_hCam      = NULL;       // Camera handle
static CAM_STRM_HANDLE  s_hStrm     = NULL;       // Stream handle
static HANDLE           s_hStrmEvt  = NULL;       // Completion event for stream.

static CAM_PIXEL_FORMAT s_pixelFormat;

//#define USE_IPL_IMAGE

#ifdef USE_IPL_IMAGE
static ::IplImage       *s_pIplImage = NULL;
#else
static cv::Mat          s_openCvImage;
#endif

static char             s_szWindowTitle[] = "OpenCV Test";

/*****************************************************************************/
/* Functions                                                                 */
/*****************************************************************************/
int _tmain(int argc, _TCHAR* argv[])
{
    uint32_t  uiStatus = 0;

    do
    {
        uiStatus = Initialize();
```



```
        if (uiStatus != 0)
        {
            printf("Failed Initialize(%d), Status=0x%08X.\n", uiStatus, s_uiStatus);
            break;
        }

        // Set software one-shot trigger mode.
        uiStatus = SetTriggerMode();
        if (uiStatus != 0)
        {
            printf("Failed SetTriggerMode(%d), Status=0x%08X.\n", uiStatus, s_uiStatus);
            break;
        }

        // Open stream.
        uiStatus = OpenStream();
        if (uiStatus != 0)
        {
            printf("Failed OpenStream(%d), Status=0x%08X.\n", uiStatus, s_uiStatus);
            break;
        }

        // Start acquisition
        uiStatus = StartAcquisition();
        if (uiStatus != 0)
        {
            printf("Failed StartAcquisition(%d), Status=0x%08X.\n", uiStatus, s_uiStatus);
            break;
        }

        // Receive images.
        uiStatus = ImageHandling();
        if (uiStatus != 0)
        {
            printf("Failed ImageHandling(%d), Status=0x%08X.\n", uiStatus, s_uiStatus);
        }

        // Stop acquisition.
        uiStatus = StopAcquisition();
        if(uiStatus != 0)
        {
            printf("Failed StopAcquisition(%d), Status=0x%08X.\n", uiStatus, s_uiStatus);
            break;
        }
    } while(0);

    // Close stream.
    CloseStream();

    // Terminate.
    Terminate();

    return 0;
}

uint32_t Initialize()
{
    uint32_t  uiNum;

    // API initialization.
    s_uiStatus = Sys_Initialize();
    if (s_uiStatus != CAM_API_STS_SUCCESS)
        return 1;

    // Get uiNumber of camera.
    s_uiStatus = Sys_GetNumOfCameras(&uiNum);
```

```
    if (s_uiStatus != CAM_API_STS_SUCCESS)
        return 3;

    if (uiNum == 0)
    {
         printf(" No cameras found.\n");
        return 4;
    }
    printf(" %d camera(s) found.\n", uiNum);

    // Open camera that is detected first, in this sample code.
    uint32_t iCamNo = 0;
    s_uiStatus = Cam_Open(iCamNo, &s_hCam);

    if (s_uiStatus != CAM_API_STS_SUCCESS)
        return 6;

    return 0;
}

void Terminate()
{
    // Close camera.
    if (s_hCam != NULL)
        Cam_Close(s_hCam);

    // Terminate system.
    Sys_Terminate();
}

uint32_t SetTriggerMode()
{
    // Set TriggerMode false, in this sample code.
    s_uiStatus = SetCamTriggerMode(s_hCam, false);
    if (s_uiStatus != CAM_API_STS_SUCCESS)
        return 40;

    return 0;
}

uint32_t OpenStream()
{
    // Create completion event for stream.
    s_hStrmEvt = CreateEvent(NULL, FALSE, FALSE, NULL);
    if (s_hStrmEvt == NULL)
        return 80;

    // Open stream channel.
     s_uiStatus = Strm_OpenSimple(s_hCam, &s_hStrm, &s_uiImgBufSize, s_hStrmEvt);
    if (s_uiStatus != CAM_API_STS_SUCCESS)
        return 81;

    uint32_t uiWidth, uiHeight;

    s_uiStatus = GetCamWidth(s_hCam, &uiWidth);
    if (s_uiStatus != CAM_API_STS_SUCCESS)
        return 900;

     s_uiStatus = GetCamHeight(s_hCam, &uiHeight);
    if (s_uiStatus != CAM_API_STS_SUCCESS)
        return 901;

    s_uiStatus = GetCamPixelFormat(s_hCam, &s_pixelFormat);
    if (s_uiStatus != CAM_API_STS_SUCCESS)
        return 902;
```

```
    if (s_pixelFormat == PXL_FMT_Mono8)
    {
#ifdef USE_IPL_IMAGE
        s_pIplImage = ::cvCreateImage( ::cvSize(uiWidth,uiHeight), IPL_DEPTH_8U, 1);
#else
        s_openCvImage = cv::Mat(cv::Size(uiWidth, uiHeight), CV_8UC1, cv::Scalar::all(0));
#endif
    } else
    {
        // Allocate image bufer for receiving image data from TeliCamAPI.
        s_pucImgBuf = (uint8_t *)VirtualAlloc(NULL,
                                        s_uiImgBufSize,
                                        MEM_RESERVE | MEM_COMMIT,
                                        PAGE_EXECUTE_READWRITE);
        if (s_pucImgBuf == NULL)
            return 82;

#ifdef USE_IPL_IMAGE
        s_pIplImage = ::cvCreateImage( ::cvSize(uiWidth,uiHeight), IPL_DEPTH_8U, 3);
#else
        s_openCvImage = cv::Mat(cv::Size(uiWidth, uiHeight), CV_8UC3, cv::Scalar::all(0));
#endif
    }

    cv::namedWindow(s_szWindowTitle, CV_WINDOW_AUTOSIZE);

    return 0;
}

uint32_t CloseStream()
{
    // Close stream.
    if (s_hStrm != NULL)
        Strm_Close(s_hStrm);

    // Close completion event for stream.
    if (s_hStrmEvt != NULL)
        CloseHandle(s_hStrmEvt);

    // Release image buffer.
    if (s_pucImgBuf != NULL)
        VirtualFree(s_pucImgBuf, 0, MEM_RELEASE);

#ifdef USE_IPL_IMAGE
    if (s_pIplImage != NULL)
        ::cvReleaseImage( &s_pIplImage);
#else
    // Do Nothing
#endif

    cv::destroyWindow(s_szWindowTitle);

    return 0;
}

uint32_t StartAcquisition()
{
    s_uiStatus = Strm_Start(s_hStrm);
    if (s_uiStatus != CAM_API_STS_SUCCESS)
        return 100;

    return 0;
}

uint32_t StopAcquisition()
{
```

```
    s_uiStatus = Strm_Stop(s_hStrm);
    if (s_uiStatus != CAM_API_STS_SUCCESS)
        return 120;

    return 0;
}

uint32_t ImageHandling()
{
    const uint32_t  uiTimeout = 5000;    // 5sec.
    uint32_t        uiRes;
    uint32_t        uiSize;
    CAM_IMAGE_INFO  imageInfo;
    int             key;

    while(1)
    {
        // Wait for receiving image completion event.
        uiRes = WaitForSingleObject(s_hStrmEvt, uiTimeout);
        if (uiRes != WAIT_OBJECT_0)
            return 182;     // Timeout.

        if (s_pixelFormat == PXL_FMT_Mono8)
        {
#ifdef USE_IPL_IMAGE
            s_uiStatus = Strm_ReadCurrentImage(s_hStrm, s_pIplImage->imageData, &uiSize,
&imageInfo);
            if (s_uiStatus != CAM_API_STS_SUCCESS)
                return 183;

            ::cvShowImage(s_szWindowTitle, s_pIplImage);
#else
            s_uiStatus  =  Strm_ReadCurrentImage(s_hStrm,  s_openCvImage.data,  &uiSize,
&imageInfo);
            if (s_uiStatus != CAM_API_STS_SUCCESS)
                return 183;

            cv::imshow(s_szWindowTitle, s_openCvImage);
#endif
        } else
        {
            // Copy received image data to local buffer, and get size of received image.
            s_uiStatus = Strm_ReadCurrentImage(s_hStrm, s_pucImgBuf, &uiSize, &imageInfo);
            if (s_uiStatus != CAM_API_STS_SUCCESS)
                return 183;

            // Convert image format.
#ifdef USE_IPL_IMAGE
            s_uiStatus = ConvImage(DST_FMT_BGR24, imageInfo.uiPixelFormat, true,
                s_pIplImage->imageData, s_pucImgBuf, imageInfo.uiSizeX, imageInfo.uiSizeY);
            if (s_uiStatus != CAM_API_STS_SUCCESS)
                return 910;

            ::cvShowImage(s_szWindowTitle, s_pIplImage);
#else
            s_uiStatus = ConvImage(DST_FMT_BGR24, imageInfo.uiPixelFormat, true,
                s_openCvImage.data, s_pucImgBuf, imageInfo.uiSizeX, imageInfo.uiSizeY);
            if (s_uiStatus != CAM_API_STS_SUCCESS)
                return 910;

            cv::imshow(s_szWindowTitle, s_openCvImage);
#endif
        }

        key = cv::waitKey(10);
        if (key == 0x1b)    // [ESC] key
```

```
            break;
    }

    return 0;
}
```

End of document in Japanese

4. Others

4.1. Revision History

Date	Version	Description
2017/06/21	1.0.0	Created the initial version

4.2. Disclaimer

The disclaimer of this document including example code is described in “License Agreement TeliCamSDK Eng.pdf” in TeliCamSDK installation folder.
Make sure to read this Agreement carefully before using it.
Refer to TeliCamSDK installation folder/Documents/License folder

4.3. 8.3. License

Microsoft and Visual C++ are trade mark or registered trade mark of Microsoft Corporation
GigE Vision™ is standard by AIA(Automated Imaging Association).
USB3 Vision™ is standard by AIA(Automated Imaging Association)
GenICam™ is registered trade mark of EMVA(European Machine Vision association)
Other product names in this document are trade mark or registered trade mark of these companies.

#5400-0384