USB Camera for Robot Vision
From the installation to the pixels

Pierre Lamon & Illah Nourbakhsh, EPFL and The Robotics Institute

1. Introduction

These pages describe how to connect an USB camera to your PC and how to grab pictures from. First the hardware is described and next the software part is considered.

2. The hardware

2.1 The USB camera

The USB camera used is the QuickCam Pro from Logitech. It is a cheap color CCD camera with true 640 x 480 resolution. The focus can be set manually and the orientation of the camera is easy to change. Look at the Logitech web site for further information: **Error! Bookmark not defined.**.

2.2 The PCMCIA card

If your notebook doesn't have an USB port you should install a PCMCIA card. The card[1] used provides two USB ports and requires Windows 98 and a Card bus PC Card Slot type II. To install the device insert the card, power on your notebook and follow instruction provided by Windows 98. For more information about the product please check the ADS Technologies web site: http://www.adstech.com/products/usb_port_cardbus.html.

This device is compatible for the following notebooks:

Compaq: Armada 1598MT, 4220T, 7380DMT, 7792DMT
Compaq: Presario 1625,1626,1640
CTX: EzBook 777MT-MK
Digital HiNote Ultra: 2000 GTX5266M
Fujitsu: Lifebook 555Tx
Hitachi: Visionbook 7755
IBM: Thinkpad 765L
Micron: Transport XPE Series
Sony: VAIO PCG-800

Other models may be compatible. You should check your notebook for available type II Card Bus slots.

[1] The full order ID is "PCMCIA to USB card USBX-501"

2.3 Installing the driver for the camera

The camera is sold with instructions and a CD-ROM with the drivers and different programs. Please check the web site in section 2.1 if you want to download the latest updates of the drivers. They exist for Windows 95/98 and NT 4.0. The setup program will copy files on your hard drive, install drivers and register some ActiveX controls in the system.

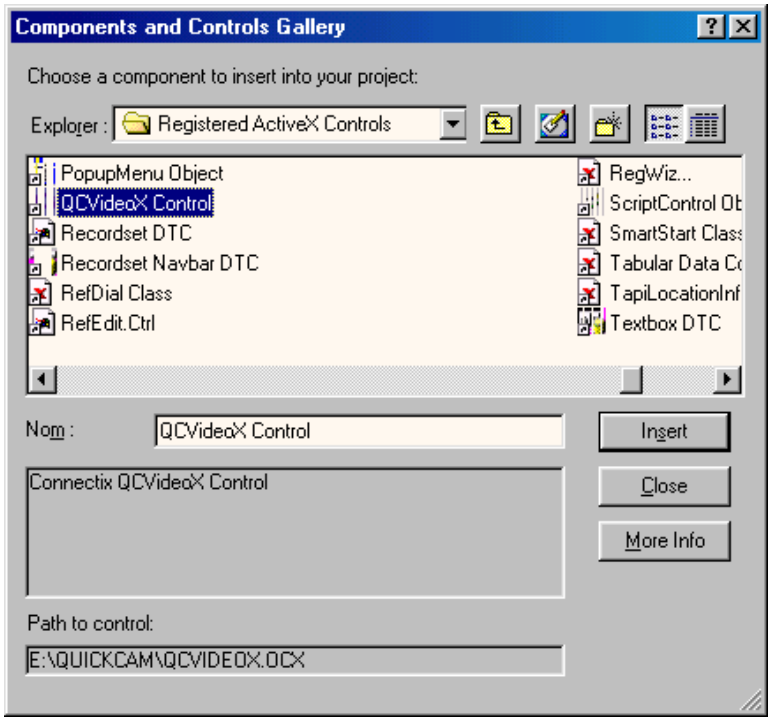
2.4 How to grab frames from a Visual C++ application

The following example will describe how to grab frames and access pixels from a Visual C++ 6.0 executable. The interface with the camera is done through an ActiveX control installed by the previous setup program. A demo program named "USBCam.exe" and the source code is available as example (the name of the project is "USBCam.dsw").

2.4.1 Insert the ActiveX Control in a Visual C++ project

After you created a new project in Visual C++ you have to insert the ActiveX control. For that follow the instructions below:

- Click on "Project" in the main menu
- Display the sub menu "Add To Project"
- Click on the item "Components and Controls…". The following window will appear.



- Select the control “QCVideoX Control” and press the button “Insert”. This will create the class CQCVideoX in your project and generate the files qcvideox.h and qcvideox.cpp.

3.2.2 Initialize camera and grab frames

Now the control is available in the dialog toolbar. Drag the control on your dialog and create a variable associated to the control with ClassWizzard. All the functions of the ActiveX control are available now by including the file “qcvideox.h” in code files. The following code shows how to initialize the USB camera and save a frame on the hard drive. The variable corresponding to the control is named m_VideoFrame.

```
…
#include “qcvideox.h”
…

// Initialize the camera. The parameter is the ID of the camera
m_VideoFrame.Initialization (0);

// Set color mode 0 = manual, 1 = automatic
m_VideoFrame.SetColorMode (0);

// If ColorMode = 0 next sensitivities can be set manually (0-255)
m_VideoFrame.SetCyanRedColor (127);
m_VideoFrame.SetYellowBlueColor (88);

// Set brightness mode 0 = manual, 1 = automatic
m_VideoFrame.SetBrightnessMode (0);

// If BrightnessMode = 0 next sensitivities can be set manually (0-255)
m_VideoFrame.SetExposure (128);
m_VideoFrame.SetBrightnessMagnitude (150);
m_VideoFrame.SetGain (23);

// Set contrast and saturation
m_VideoFrame.SetContrastMagnitude (160);
m_VideoFrame.SetSaturation  (128);

// Set orientation 0 = normal, 1 = mirror, 2 = flip, 3 = flip + mirror
m_VideoFrame.SetOrientation (0);

// Set PictureSmart 0 = off, 1 = on (correct distortion of the image if on)
m_VideoFrame.SetPictureSmart (0);

// Set Low Light Filter 0 = off, 1 = on
m_VideoFrame.SetLowLightFilter (0);

// Set the quality of the image (0-4; 4 best quality)
m_VideoFrame.SetCompression (4);

// Save image on the hard drive
```

```
m_VideoFrame.SaveSingleFrameToFile (filename);

// Un-initialize the camera
m_VideoFrame.UnInitialization();
```
Please *look* at the file *"qcvideox.h" for more functions*.

2.5 How to access pixels

The class CImage provides functions that make the manipulation of bitmaps easier. The example below describes how to apply a 3x3 filter on an image (Picture0.bmp) and how to access pixels.

```
…
#include "Image.h"
…
typedef short MATRIX_3X3[3][3];

CString filename = "Picture0.bmp";
CClientDC        *pDC;
CImage SrcImage;
CImage *pDestImage;
MATRIX_3X3 pMat3x3 ={{-1,0,1},{-2,0,2},{-1,0,1}};

// Read the file, init and alloc memory for the source image
SrcImage.ReadFromFile (filename);

// Display all information about the image
SrcImage.DisplayInfo ();

// Access pixels
SrcImage.SetPixel (20,   // Column index
                   20,   // Line index
                   255, // Value of red
                   0,    // Value of green
                   0 );  // Value of blue


// Access directly the array of bytes: Pixel[LineNumber][IndexOfTheByte]
// Example:  access the green value of the pixel on the 201 th line and first column
Pixel[200][1] = 128;

// Create the destination image in memory (same size of SrcImage)
pDestImage = new CImage (SrcImage.pHead->biWidth,        // Width of the image
                         SrcImage.pHead->biHeight,       // Height of the image
                         SrcImage. bPerPixel );          // Bytes per pixel


// Apply a 3x3 filter on the source image
```

```
SrcImage.Filter_3x3 (pDestImage,            // Destination image
                     RG_CHANNEL,      // Channel to apply the filter on (see header)
                     FALSE,      // TRUE if you want to use Sobel matrix (vertical)
                                 // Speed optimized
                     pMat3x3 );  // Pointer on a 3x3 matrix filter

// Initialization of the screen device context
pDC = new CClientDC(this);

// Display the image on the screen
pDestImage.DisplayImage(pDC->m_hDC,      // Device context
                        100,             // x coordinate (screen)
                        20,              // y coordinate (screen)
                        pDestImage.pHead->biWidth,        // Width of the source
                        pDestImage.pHead->biHeigth,       // Heigth of the source
                        0,               // first column to be displayed
                        ALL_IMAGE,       // Mode (see header file)
                        SRCCOPY );       // Mask (see CBitmap BitBlt function help)


// Convert pixels from RGB to HSI
SrcImage.HSI_Values (pDestImage);

// Stores the image on the harddrive
pDestImage->StoreInFile (filename);

// Frees the image
delete pDestImage;
…
```

The reader is invited to have a look at the files “Image.h” and “Image.cpp” for more functions and information.