ViSP 2.9.0: Visual Servoing Platform

Getting Started for Windows

Lagadic project
http://www.irisa.fr/lagadic

February 18, 2014

François Chaumette
Eric Marchand
Nicolas Melchior
Fabien Spindler

INVENTORS FOR THE DIGITAL WORLD

Contents

This getting started for Windows is for those who want to use ViSP under Windows and who do not intend to participate in the development of the library. Its goal is to help them to start writing a program using ViSP as a third party library without going into details. It does not replace the ViSP source code documentation which can be consulted on the website:

http://www.irisa.fr/lagadic/visp

Thereafter, the different steps between the download and the use of the library will be described.

1 Introduction

Before to download ViSP and try to build it, it is advised to install an IDE (Visual C++, Borland, Eclipse, . . .) and the last version of CMake which can be found at the address : http://www.cmake.org.

As described in section 5.1, some ViSP capabilities require third party libraries to be installed. But they are not required to build ViSP. Don't worry, if you use a function which requires another library you do not have installed yet, you will be warned during the execution of your own program.

2 Where and how downloading ViSP

First you have to know that there are two ways to download ViSP source code. The simplest one consists in downloading the zip file which can be found at the address:

http://www.irisa.fr/lagadic/visp/download.html

It contains a release version of the source code. Unzip the package in the folder of your choice and go directly to Section 3.

The other way to recover ViSP is to download it from Subversion repository hosted on InriaGForge http://gforge.inria.fr/projects/visp/. Subversion is a tool for a team of developers which enable to manage the source code during the development process. The advantage is that you can have the current development version of the code. The drawback is that it is not necessary stable and the last functions could be not documented yet. Prior to download something from Subversion repository you have to install a Subversion client like TortoiseSVN, either Slik Subversion or Subversion package from Cygwin. Then you can use the following address to recover ViSP by checking out the source code files:

svn://scm.gforge.inria.fr/svn/visp/trunk/ViSP

Regardless the method you used to download ViSP, you have now a version of the source code which must be build to be used.

3 How to build ViSP under Windows

Now, the step consists in preparing the build by creating a project or a makefile depending on your IDE. It will be done thanks to CMake.

1. Execute CMake to get the GUI presented Figure 1.

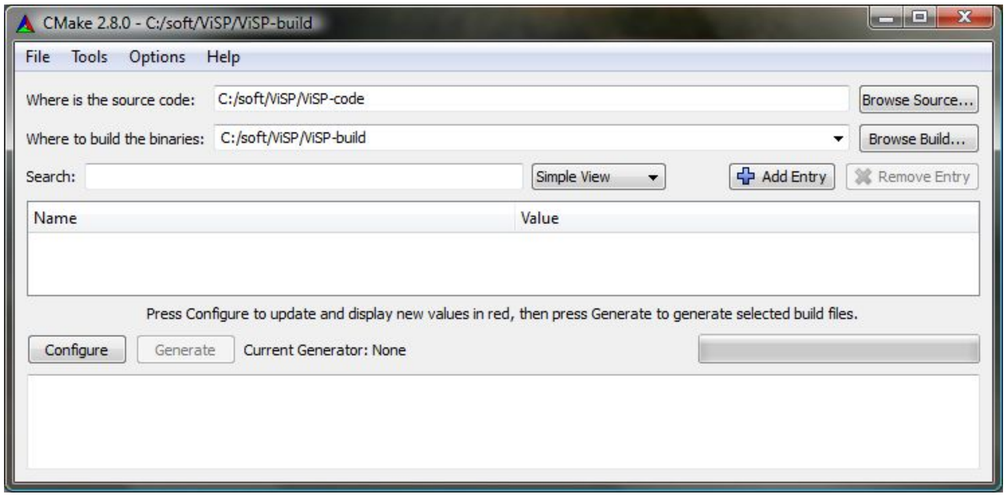
2. In the first box "*Where is the source code*" (see Fig. 1) you have to set the path to the folder which contains the ViSP source code.

3. In the second box "*Where to build the binaries*" (see Fig. 1) set the path to the folder you choose to contain ViSP binaries obtained after the build stage described at step 6. If it doesn't exist yet, it will be created automatically. Let us denote this path as `VISP_BUILD_DIR`.

   An advice is to choose two different folders, one for the source code and another one for the build version. There are two advantages to do this. Firstly, the folder which contains the source code will not be contaminated by files created by CMake. So, if you want to modify ViSP it will be easier. Secondly, it allows to have more than one build version of ViSP. Indeed, it exists numerous possibilities to build it depending on the third party libraries you are using. So you could use the version which matches the best to your own project.

4. Click on the "*Configure*" button (see Fig. 1). The IDE you want to use will be asked. You have to indicate it in order to enable CMake to create the right project configuration files. Continue to click on "*Configure*" until the "*Generate*" button becomes active. During this step you are allowed to modify any options. Figure 2 shows for example how to print advanced options that can be modified like the `CMAKE_INSTALL_PREFIX` variable used to specify where ViSP will be installed.

5. After clicking on "*Configure*", you will be allowed to click on "*Generate*" (see Fig. 2). CMake will create the useful configuration files in the folder you indicate in the box "*Where to build the binaries*". Now you are allowed to build ViSP binaries corresponding to the library and the examples.



Where is the source code: C:/soft/ViSP/ViSP-code
Where to build the binaries: C:/soft/ViSP/ViSP-build
Search:

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: None

Figure 1: CMake GUI obtained with CMake 2.8 that allows to configure ViSP on your computer.

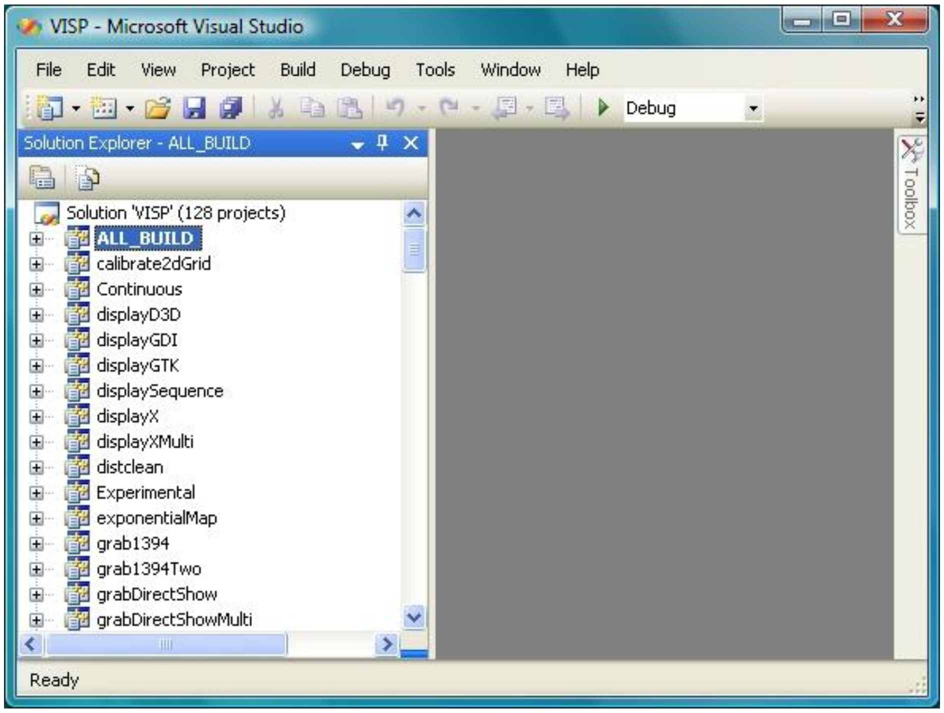
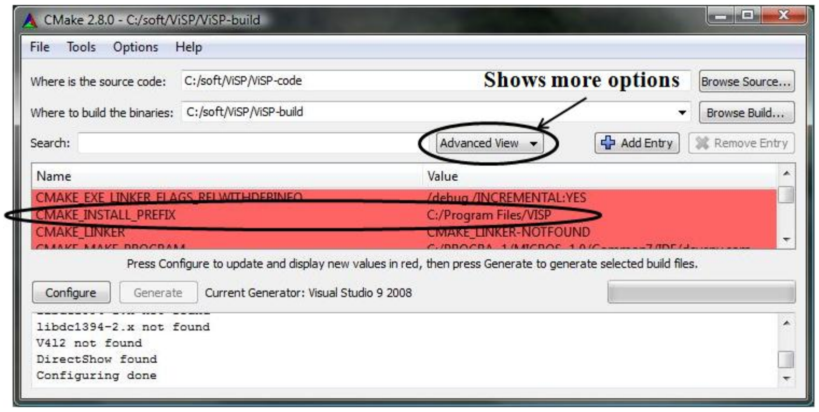


Figure 2: This CMake snapshot shows where to set the `CMAKE_INSTALL_PREFIX` variable specifying the directory where ViSP will be installed after the build. By default, this variable is set to `C:/Program Files/VISP`. As shown Fig. 10, this snapshot shows also the location of the menu that allows to print more advanced options like the location of third party headers and libraries.

6. Open the project file which has been created by CMake. For example if you are using Microsoft Visual Studio 2008, its name is `VISP.sln`. The screenshot Fig. 3 comes from this IDE but you will find similar things with others.



Where is the source code: C:/soft/ViSP/ViSP-code

Where to build the binaries: C:/soft/ViSP/ViSP-build

Search:

Name	Value
CMAKE_EXE_LINKER_FLAGS_RELWITHDEBINFO	/debug /INCREMENTAL:YES
CMAKE_INSTALL_PREFIX	C:/Program Files/VISP
CMAKE_LINKER	CMAKE_LINKER-NOTFOUND

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Visual Studio 9 2008

```
libdc1394-2.x not found
V4l2 not found
DirectShow found
Configuring done
```

Figure 3: `VISP.sln` Visual Studio solution file created by CMake opened with Microsoft Visual Studio 2008. To build ViSP library and all the examples select the `ALL_BUILD` project. To install ViSP headers and library in the directory corresponding to the `CMAKE_INSTALL_PREFIX` variable select and build the `INSTALL` project. To build the HTML documentation select and build the `html-doc` project.

7. The last thing you have to do is to build the `ALL_BUILD` project (see Fig. 3). Thus the library will be created and the examples will be compiled. The binaries will be contained in the folder you indicate in the box "*Where to build the binaries*". You can also build the `INSTALL` project which will install the ViSP headers and library in the path corresponding to the `CMAKE_INSTALL_PREFIX` (by default set to C:/Program Files/VISP). You are allowed to modify this installation path during step 4. In order to produce the HTML documentation with Doxygen, you can build the `html-doc` project. The documentation entry point is than `VISP_BUILD_DIR/doc/html/index.html`.

4 How to use ViSP as a third party library under Windows

4.1 How to create a HelloWorld project using ViSP with CMake

In this section you will learn how to create a HelloWorld Visual Studio project using ViSP as a third party library. This step is very simple if you still use CMake to configure your project.

1. First you have to create a folder where you want to put the HelloWorld project.

2. Then create inside this folder the `HelloWorld.cpp` file you want to build and a text file called `CMakeLists.txt` that corresponds to the HelloWorld configuration file that will be used by CMake. The following simple example shows you to fill in these files [1].

   `HelloWorld.cpp`:

[1] `HelloWorld.cpp` and `CMakeLists.txt` files are available in ViSP source tree in `example/manual/hello-world/CMake` directory.

```cpp
#include <iostream>

#include <visp/vpDebug.h>
#include <visp/vpImage.h>
#include <visp/vpImageIo.h>

int main()
{
  std::cout << "ViSP Hello World example" <<std::endl;

  vpImage<unsigned char> I(288, 384);

  I = 128;

  std::cout << "ViSP creates \"./myimage.pgm\" B&W image " <<std::endl;
  vpImageIo::write(I, "./myimage.pgm");

  return 0;
}
```

`CMakeLists.txt` :

```cmake
PROJECT(HelloWorld)

CMAKE_MINIMUM_REQUIRED(VERSION 2.6)

FIND_PACKAGE(VISP REQUIRED)
IF(VISP_FOUND)
  INCLUDE(${VISP_USE_FILE})
ENDIF(VISP_FOUND)

ADD_EXECUTABLE(HelloWorld HelloWorld.cpp)
```

3. Then, start CMake to configure the HelloWorld project (see Fig. 4).

4. In the first box "*Where is the source code*" (see Fig. 4), indicate the path to your source code (ie the path to the folder you created in step 1).

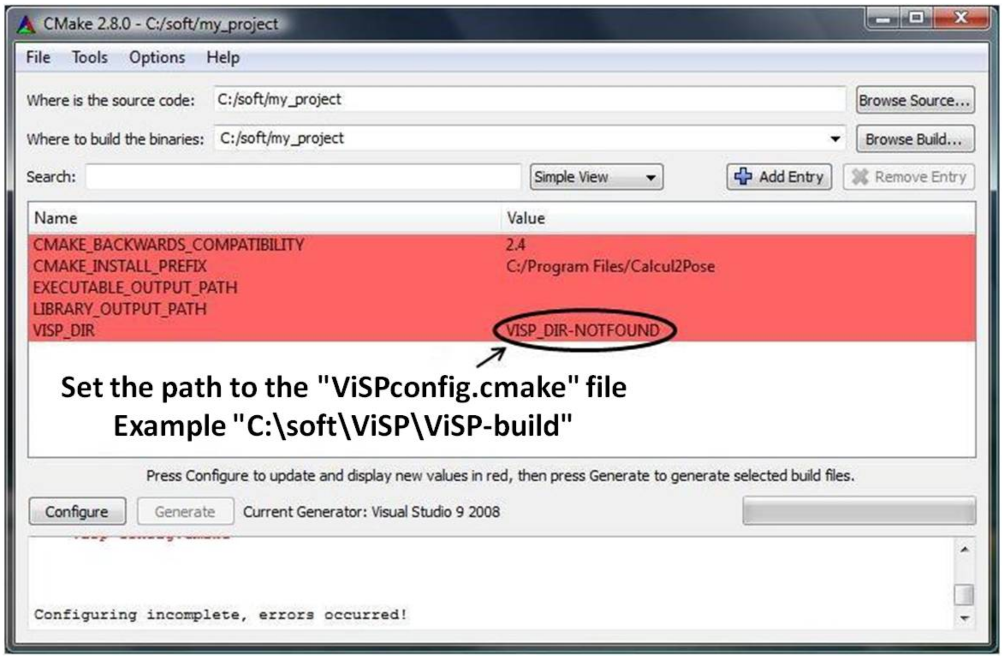
5. In the second box "*Where to build the binaries*" (see Fig. 4), indicate where you want to build the binaries corresponding to your project.

6. You can now click on "*Configure*" (see Fig. 4).

   If CMake says that it can't find the ViSP library (`VISP_DIR` variable is than set to `VISP_DIR-NOTFOUND`, see Fig. 4), you may indicate the path to the folder containing a build version of ViSP. More precisely, you have to give the path to the `ViSPConfig.cmake` file. Typically, you can find it in `VISP_BUILD_DIR` (replace `VISP_BUILD_DIR` with the path to the folder where you build ViSP, see Section 3, step 3). If you install ViSP (see Section 3, step 7), you can also set `VISP_DIR` to the following path `C:/Program Files/VISP`.

7. Then click on "*Configure*" until having the right to click on "*Generate*" button (see Fig. 4). It is the same as for ViSP. CMake will create a project for your own IDE. You need now to download the project (in our case the `HelloWorld.sln` Visual Studio solution file) in your IDE to build the binaries.

The advantage to use CMake is that all the links are automatically done and especially with the third party libraries on which ViSP is depending.



Where is the source code: C:/soft/my_project
Where to build the binaries: C:/soft/my_project

Name	Value
CMAKE_BACKWARDS_COMPATIBILITY	2.4
CMAKE_INSTALL_PREFIX	C:/Program Files/Calcul2Pose
EXECUTABLE_OUTPUT_PATH	
LIBRARY_OUTPUT_PATH	
VISP_DIR	VISP_DIR-NOTFOUND

Set the path to the "ViSPconfig.cmake" file
Example "C:\soft\ViSP\ViSP-build"

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Visual Studio 9 2008

```
Configuring incomplete, errors occurred!
```

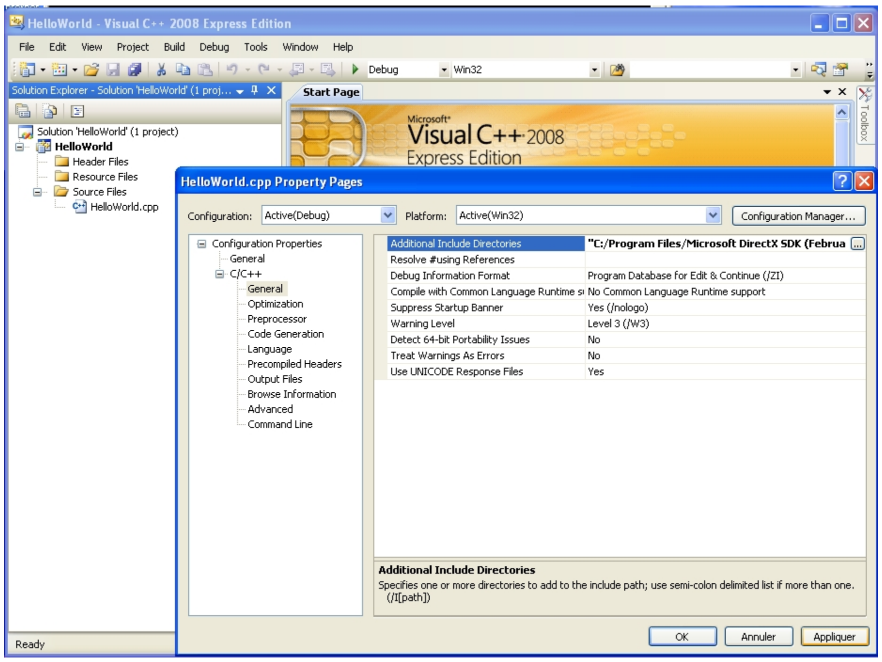
Figure 4: CMake based configuration of the HelloWorld project that uses ViSP as a third party library.

4.2 How to create a HelloWorld project using ViSP without CMake

It is also possible to develop your project using ViSP as a third party library without the help of CMake. In such a case, you have to set the properties of your project with additional include directories, preprocessor definitions, language definition, library directories and library dependencies. In order to help users, ViSP provides a `visp-config.bat` batch file that may give the additional properties values. This file is produced during CMake configuration (see Section 3, step 4) and is located in `VISP_BUILD_DIR/bin` directory. If you install ViSP (see Section 3, step 7), you will also found this file in `C:/Program Files/VISP/bin`. The usage of this batch file is given by "`visp-config.bat --help`" executed in a DOS command window. The following steps explain how to set the properties of a HelloWorld project using the `visp-config.bat` outputs. Once your project is created, with Visual Studio you may edit its properties.

1. Setting additional include directories: The first thing you have to do is to indicate the folders containing the headers belonging to ViSP and the third party libraries used to build ViSP (see Fig. 5). Note that the include directory for the ViSP library is "`VISP_BUILD_DIR/include`". If you install ViSP (see Section 3, step 7), the include directory for the ViSP library is rather `C:/Program Files/VISP/include`. To get all the additional include directories you can copy [2] the result of the "`visp-build.bat --include`" command executed in a DOS command window and paste it in the box "*Configuration Properties→C/C++→General→ Additional include directories*". Note that the result of this command contains already ViSP include directory.

[2]**Tip:** To copy/past the output of the `visp-build.bat` command we suggest to redirect the output in a text file by executing for example "`visp-build.bat --include > include.txt`". The resulting `include.txt` file can than be opened in the Wordpad from which the copy/past can be done easily.



Configuration: Active(Debug) Platform: Active(Win32)

Additional Include Directories	"C:/Program Files/Microsoft DirectX SDK (Februa
Resolve #using References	
Debug Information Format	Program Database for Edit & Continue (/ZI)
Compile with Common Language Runtime s	No Common Language Runtime support
Suppress Startup Banner	Yes (/nologo)
Warning Level	Level 3 (/W3)
Detect 64-bit Portability Issues	No
Treat Warnings As Errors	No
Use UNICODE Response Files	Yes

Additional Include Directories
Specifies one or more directories to add to the include path; use semi-colon delimited list if more than one. (/I[path])

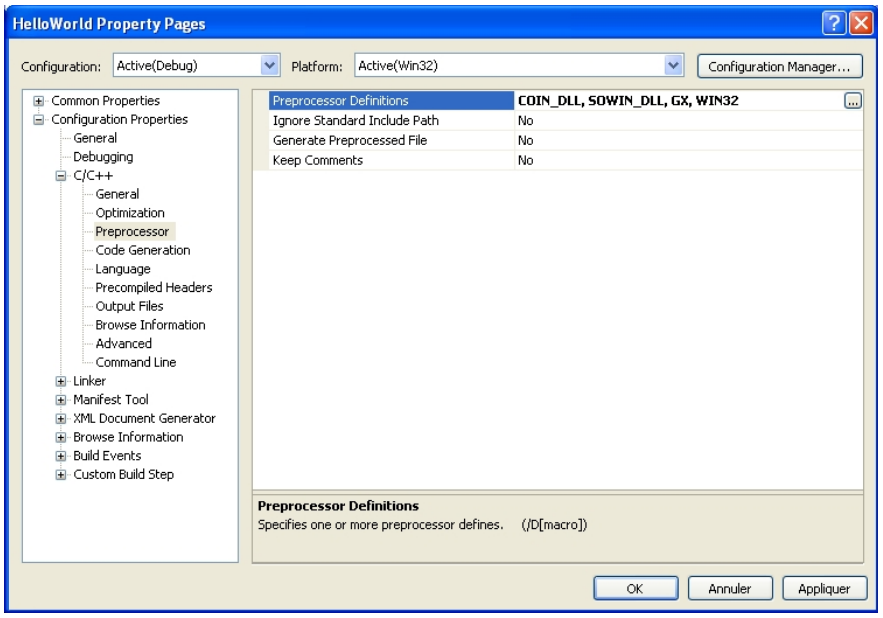
Figure 5: This Visual Studio snapshot shows where to set the additional include directories requested to build the HelloWorld project using ViSP as a third party library. The list of additional include directories may be obtained by executing "`visp-config.bat --include`" in a DOS command window.

2. Setting additional preprocessor definitions: At least you need to add the `WIN32` preprocessor definition. To get all the requested additional preprocessor definitions you can copy[1] the result of "`visp-build.bat --def`" command executed in a DOS command window and paste it in the box "*Configuration Properties→C/C++→Preprocessor→Preprocessor Definitions*" as shown Figure 6. Note that the result given by the .bat file contains already the `WIN32` preprocessor definition.

3. Setting OpenMP support: Since ViSP-2.6.2 ViSP may support OpenMP parallelization. If /visp/ was built with OpenMP support, than you need also to activate OpenMP support to build the HelloWorld project.

4. Setting additional library directories: To set all the additional library directories you can copy[1] the result of "`visp-build.bat --libpath`" command executed in a DOS command window and paste it in the box "*Configuration Properties→Linker→General→Additional Library Directories*" as shown Figure 7. Note that ViSP library path is "`VISP_BUILD_DIR/lib/${Outdir}`" with `${Outdir}` equal to `Debug` or `Release` depending on your HelloWorld configuration. If you install ViSP (see Section 3, step 7), ViSP library path is rather `C:/Program Files/VISP/lib`. In all the cases, the result of this command contains already ViSP library directory.

5. Setting additional library dependencies : ViSP library name is `visp-2.lib`. You may also add `winmm.lib` library which is used in ViSP for time management. Depending on your configuration (debug or release), to set all the additional libraries HelloWorld depends on, you can copy[1] the result of "`visp-build.bat --libs-debug`" or "`visp-build.bat --libs-optimized`" command executed in a DOS command window and paste it in the box "*Configuration*

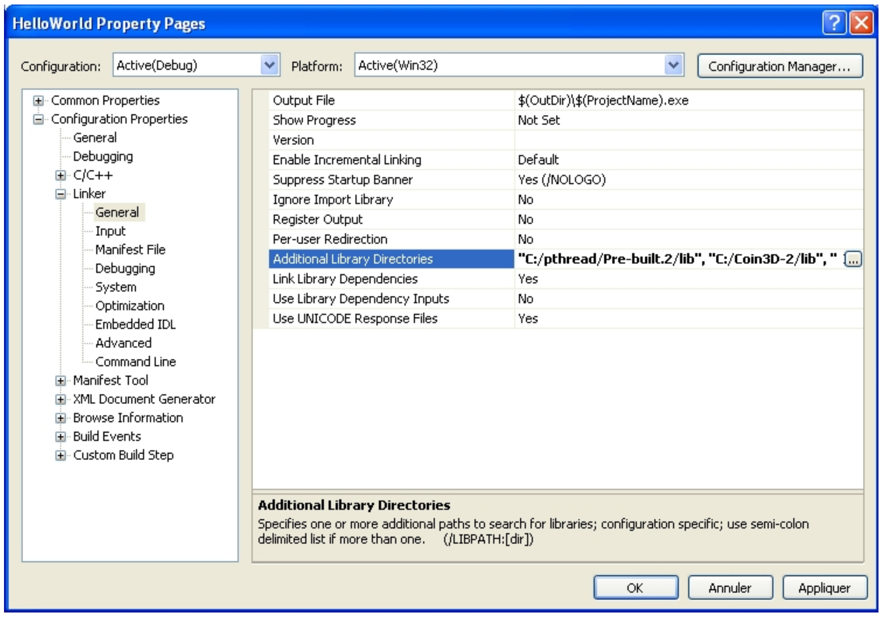


Configuration: Active(Debug) Platform: Active(Win32)

Preprocessor Definitions	COIN_DLL, SOWIN_DLL, GX, WIN32
Ignore Standard Include Path	No
Generate Preprocessed File	No
Keep Comments	No

Preprocessor Definitions
Specifies one or more preprocessor defines. (/D[macro])

Figure 6: This Visual Studio snapshot shows where to set the additional preprocessor definitions requested to build the HelloWorld project using ViSP as a third party library. The list of additional preprocessor definitions may be obtained by executing "`visp-config.bat --def`" in a DOS command window.



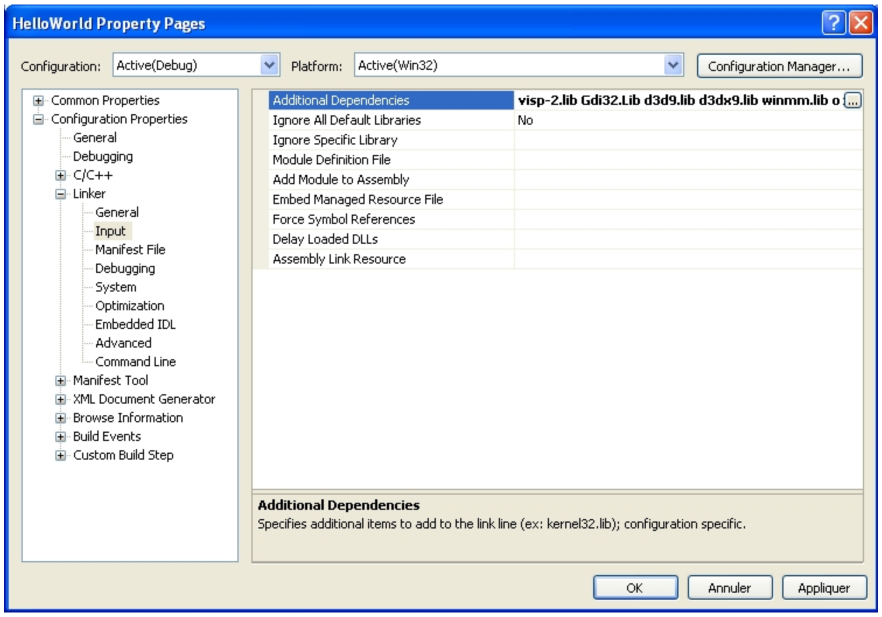
Configuration: Active(Debug) Platform: Active(Win32)

Output File	$(OutDir)\$(ProjectName).exe
Show Progress	Not Set
Version	
Enable Incremental Linking	Default
Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Per-user Redirection	No
Additional Library Directories	"C:/pthread/Pre-built.2/lib", "C:/Coin3D-2/lib", "
Link Library Dependencies	Yes
Use Library Dependency Inputs	No
Use UNICODE Response Files	Yes

Additional Library Directories
Specifies one or more additional paths to search for libraries; configuration specific; use semi-colon delimited list if more than one. (/LIBPATH:[dir])

Figure 7: This Visual Studio snapshot shows where to set the additional library directories requested to build the HelloWorld project using ViSP as a third party library. The list of additional library directories may be obtained by executing "`visp-config.bat --libpath`" in a DOS command window.

Properties→Linker→Input→Additional Dependencies" as shown Figure 8. Note that the result given by the .bat file contains already the `winmm.lib` library.



Additional Dependencies	visp-2.lib Gdi32.Lib d3d9.lib d3dx9.lib winmm.lib o
Ignore All Default Libraries	No
Ignore Specific Library	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded DLLs	
Assembly Link Resource	

Additional Dependencies
Specifies additional items to add to the link line (ex: kernel32.lib); configuration specific.

Figure 8: This Visual Studio snapshot shows where to set the additional library dependencies requested to build the HelloWorld project using ViSP as a third party library. The list of additional library dependencies may be obtained by executing "`visp-config.bat --libs`" in a DOS command window.

5 Additional information

5.1 Third party libraries used by ViSP

Many ViSP functionalities require third party libraries. This is in particular the case for simulation, framegrabbing and image viewer capabilities that require respectively Ogre 3D or Coin, SoQt and Qt, CMU 1394 or OpenCV and GTK2 or the Graphics Device Interface (GDI). If you want to know the entire list of third party libraries that can be used in ViSP you can get the information on ViSP website. Table 1 summarize these third party libraries and gives the environment variable names that could be set to help CMake to detect them.

If you are interested to know which are the third party libraries used to build ViSP on your computer, you can first check the `ViSP-third-party.txt` text file produced during the CMake configuration stage described in Section 3, step 4. This file is generated in VISP_BUILD_DIR directory. Figure 9 shows and example of such a `ViSP-third-party.txt` file content.

An other way to check which are the third party libraries that will be used while building ViSP, is to choose the "Advanced View" option in CMake GUI to have access to the CMake variables as it is shown in the screenshot Fig. 10. If you are sure you install a third party library which is noted as `NOT_FOUND`, it seems that you installed it in a not common folder. So you have the choice to set an environment variable to

ViSP capabilities	Third party library	Corresponding environment variable
Image viewer [a]	GDI	`DXSDK_DIR` or `WINSDK_DIR`
	Direct3D	`DXSDK_DIR`
	GTK2	`GTK2_DIR`
	OpenCV	`OPENCV_DIR`
SVD computation [b]	Lapack	`LAPACK_DIR`
	OpenCV	`OPENCV_DIR`
	GSL	`GSL_DIR`
Image bridges [c] and computer vision [d]	OpenCV	`OpenCV_DIR`
	Yarp	`YARP_DIR`
	Coin	`COIN_DIR` or `COINDIR`
	XML2	`XML2_DIR`
Frame grabbing [e]	CMU 1394	`CMU1394_HOME`
	OpenCV	`OPENCV_DIR`
Robots [f]	Biclops	`BICLOPS_HOME`
	Pioneer	`ARIA_HOME`
Simulator	Ogre OIS	`OGRE_HOME` and `OGRE_MEDIA_DIR` none

	Coin
	SoQt
	Qt
		`COIN_DIR` or `COINDIR`
		`COIN_DIR` or `COINDIR` or `SOQT_DIR`
		`QTDIR`

	Coin
	SoWin
		`COIN_DIR` or `COINDIR`
		`COIN_DIR` or `COINDIR` or `SOWIN_DIR`

Camera parameters parser	XML2
	iconv
		`XML2_DIR`
		`XML2_DIR` or `ICONV_DIR`

HTML documentation	Doxygen	`DOXYGEN_DIR`
	Graphviz	`GRAPHVIZ_DIR`
Image reading and writing	libjpeg	`LIBJPEG_DIR`
	libpng	`LIBPNG_DIR`

Table 1: List of environment variables that can be set throw the *Windows Control Panel* to help CMake to detect third party libraries that may be used to build ViSP.

[a]Only one device is requested to show ViSP images. GDI, Direct3D, GTK2 and OpenCV are alternatives. We suggest to use GDI which is native on Windows.

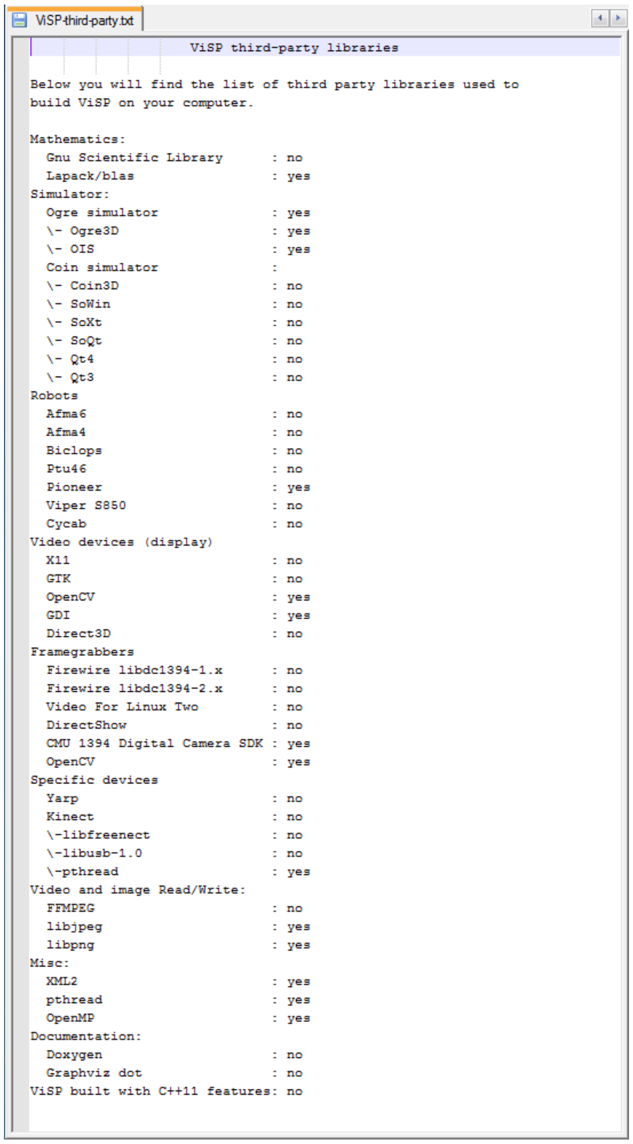
[b]To compute the pseudo inverse based on the Singular Value Decomposition (SVD) ViSP may use one of the following Lapack, OpenCV or GSL third party library. If Lapack is not found, OpenCV will be used. Then, if OpenCV is not found the Gnu Scientific Library (GSL) will be used. Finally if none of those libraries are found, an internal implementation will be used. We suggest to install Lapack.

[c]ViSP provides OpenCV and Yarp images bridges.

[d]ViSP exploit OpenCV features based for example on key points. Coin and XML2 are used by ViSP model-based tracker (MBT) to load vrml cad models of the object to track and parse tracker parameters respectively .

[e]ViSP implements wrapper over CMU 1394 able to grab images from firewire cameras and OpenCV able to handle firewire and USB cameras.

[f]These robots are interfaced with their native drivers.



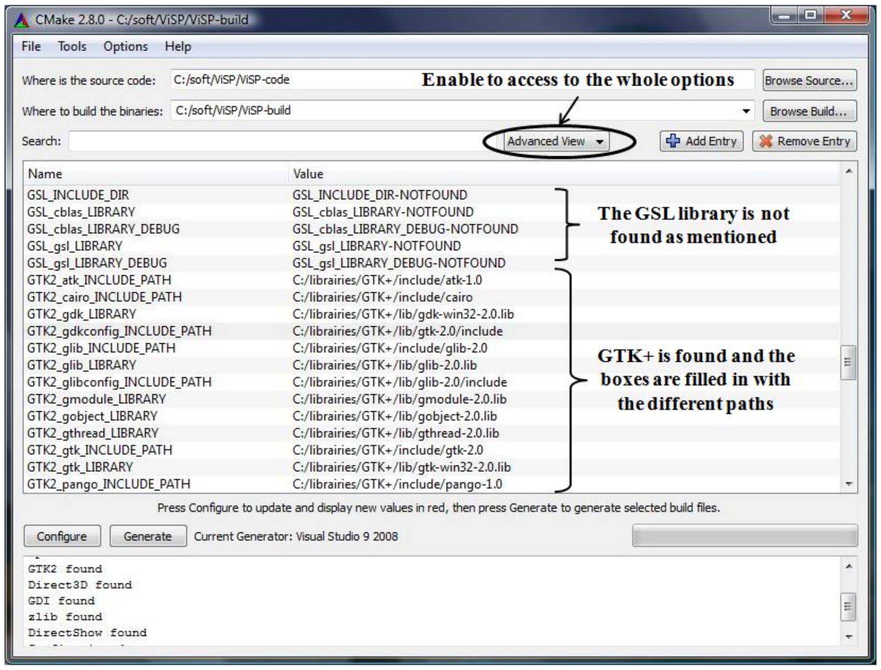
```
                    ViSP third-party libraries

Below you will find the list of third party libraries used to
build ViSP on your computer.

Mathematics:
  Gnu Scientific Library      : no
  Lapack/blas                 : yes
Simulator:
  Ogre simulator              : yes
  \- Ogre3D                   : yes
  \- OIS                      : yes
  Coin simulator              :
  \- Coin3D                   : no
  \- SoWin                    : no
  \- SoXt                     : no
  \- SoQt                     : no
  \- Qt4                      : no
  \- Qt3                      : no
Robots
  Afma6                       : no
  Afma4                       : no
  Biclops                     : no
  Ptu46                       : no
  Pioneer                     : yes
  Viper S850                  : no
  Cycab                       : no
Video devices (display)
  X11                         : no
  GTK                         : no
  OpenCV                      : yes
  GDI                         : yes
  Direct3D                    : no
Framegrabbers
  Firewire libdc1394-1.x      : no
  Firewire libdc1394-2.x      : no
  Video For Linux Two         : no
  DirectShow                  : no
  CMU 1394 Digital Camera SDK : yes
  OpenCV                      : yes
Specific devices
  Yarp                        : no
  Kinect                      : no
  \-libfreenect               : no
  \-libusb-1.0                : no
  \-pthread                   : yes
Video and image Read/Write:
  FFMPEG                      : no
  libjpeg                     : yes
  libpng                      : yes
Misc:
  XML2                        : yes
  pthread                     : yes
  OpenMP                      : yes
Documentation:
  Doxygen                     : no
  Graphviz dot                : no
ViSP built with C++11 features: no
```

Figure 9: Example of the `ViSP-third-party.txt` file content that indicates which are the third party libraries detected by CMake and used to build ViSP library on your computer.

indicate the path to the library. Table 1 gives the list of the environment variables corresponding to the third party libraries.


CMake 2.8.0 - C:/soft/ViSP/ViSP-build
File Tools Options Help
Enable to access to the whole options
Browse Source...
Browse Build...
Advanced View
Add Entry
Remove Entry
The GSL library is not found as mentioned
GTK+ is found and the boxes are filled in with the different paths
Configure
Generate

Where is the source code: C:/soft/ViSP/ViSP-code

Where to build the binaries: C:/soft/ViSP/ViSP-build

Search:

Name	Value
GSL_INCLUDE_DIR	GSL_INCLUDE_DIR-NOTFOUND
GSL_cblas_LIBRARY	GSL_cblas_LIBRARY-NOTFOUND
GSL_cblas_LIBRARY_DEBUG	GSL_cblas_LIBRARY_DEBUG-NOTFOUND
GSL_gsl_LIBRARY	GSL_gsl_LIBRARY-NOTFOUND
GSL_gsl_LIBRARY_DEBUG	GSL_gsl_LIBRARY_DEBUG-NOTFOUND
GTK2_atk_INCLUDE_PATH	C:/librairies/GTK+/include/atk-1.0
GTK2_cairo_INCLUDE_PATH	C:/librairies/GTK+/include/cairo
GTK2_gdk_LIBRARY	C:/librairies/GTK+/lib/gdk-win32-2.0.lib
GTK2_gdkconfig_INCLUDE_PATH	C:/librairies/GTK+/lib/gtk-2.0/include
GTK2_glib_INCLUDE_PATH	C:/librairies/GTK+/include/glib-2.0
GTK2_glib_LIBRARY	C:/librairies/GTK+/lib/glib-2.0.lib
GTK2_glibconfig_INCLUDE_PATH	C:/librairies/GTK+/lib/glib-2.0/include
GTK2_gmodule_LIBRARY	C:/librairies/GTK+/lib/gmodule-2.0.lib
GTK2_gobject_LIBRARY	C:/librairies/GTK+/lib/gobject-2.0.lib
GTK2_gthread_LIBRARY	C:/librairies/GTK+/lib/gthread-2.0.lib
GTK2_gtk_INCLUDE_PATH	C:/librairies/GTK+/include/gtk-2.0
GTK2_gtk_LIBRARY	C:/librairies/GTK+/lib/gtk-win32-2.0.lib
GTK2_pango_INCLUDE_PATH	C:/librairies/GTK+/include/pango-1.0

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Visual Studio 9 2008

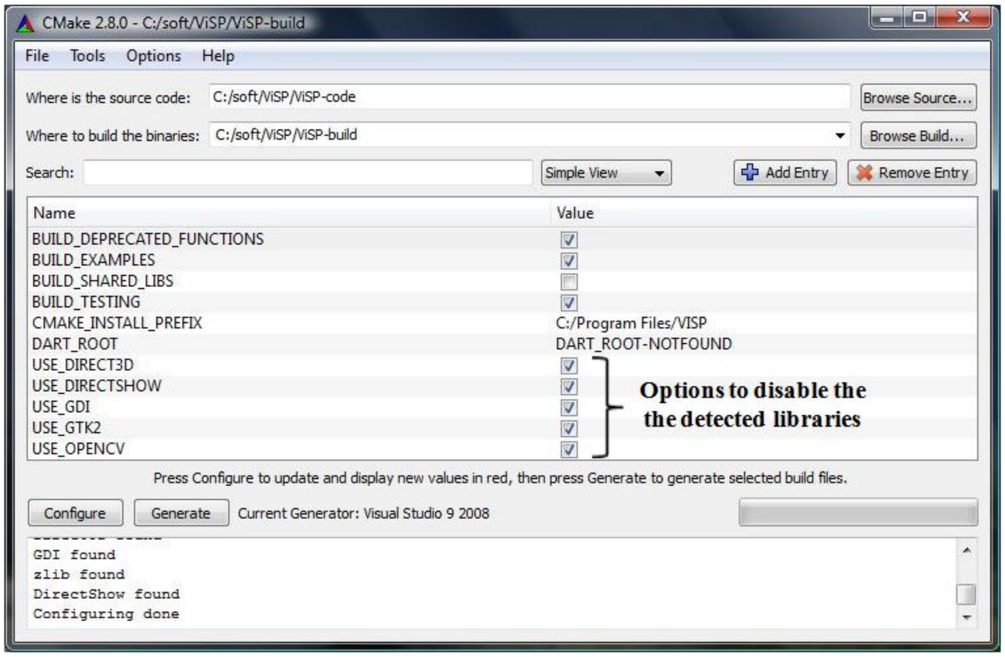
```
GTK2 found
Direct3D found
GDI found
zlib found
DirectShow found
```

Figure 10: This CMake snapshot shows how to verify if third party libraries are taken into account while building ViSP. If "Advanced View" is chosen in the menu you will get access to all CMake variables that correspond to the location of third party headers and libraries. A variable set to `NOT_FOUND` indicates that the corresponding third party library is not found.

CMake detects automatically the available third party libraries on your computer. But for some reasons, you may not want to build ViSP with all the detected libraries. You can disable these libraries during the CMake configuration (see Section 3, step 3). Indeed there are options named USE_THIRD_PARTY (see Figure 11) which appear. In that case, THIRD_PARTY is the name of the third party library which is automatically detected. To disable one of the third party library, uncheck the corresponding option (by default they are all checked).

5.2 How to execute ViSP examples

Some ViSP examples require data like images or videos as input. They can be downloaded on ViSP website. After download and unzip, you have to set the environment variable `VISP_INPUT_IMAGE_PATH` in the *Windows Control Panel*. It must be set to the parent directory containing the unzip data. For example, if you download the `ViSP-images-2.8.0.zip` file and unzip it in the folder `C:/images`, you will get a new folder named `C:/images/ViSP-images` containing the data. You need than to set the environment variable `VISP_INPUT_IMAGE_PATH` to `C:/images`. Now you should be able to execute the examples that request input data.



Where is the source code: C:/soft/ViSP/ViSP-code

Where to build the binaries: C:/soft/ViSP/ViSP-build

Search:

Name	Value
BUILD_DEPRECATED_FUNCTIONS	✓
BUILD_EXAMPLES	✓
BUILD_SHARED_LIBS	☐
BUILD_TESTING	✓
CMAKE_INSTALL_PREFIX	C:/Program Files/VISP
DART_ROOT	DART_ROOT-NOTFOUND
USE_DIRECT3D	✓
USE_DIRECTSHOW	✓
USE_GDI	✓
USE_GTK2	✓
USE_OPENCV	✓

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Visual Studio 9 2008

```
GDI found
zlib found
DirectShow found
Configuring done
```

Figure 11: This CMake snapshot shows were you can find the options used to enable or disable the detected third party libraries.