Calibration Tool - Setup Manual

Lovicourt Léo

May 13, 2022

1 Windows setup

In a first time, you will have to download the C/C++ compiler MinGW (*MingW-W64-builds* package). During installation, choose the *posix* threads and the *seh* exception. Once the installation is done, you have to add a variable to the environment variable *"Path"*.

Then, there are 2 ways for setting up every library used for the program. For both, extract the archive in a folder of your choice.

1.1 Using the files provided in the folder

There is nothing to install : all the necessary dynamic libraries (*DLL* files) and header files (*h* and *hpp* files) are already provided in the folders *libs* and *include*. However, if you have to add new functionalities, you may have to install the libraries figuring in the next section.

1.2 Installing the required libraries

You will have to install 2 libraries : OpenCV (4.X) and wxWidgets (3.X).

For OpenCV, you should not download the Windows executable file as it does not contain the *contrib* modules. Instead, download the source files and build it manually (with *CMake* for example). While building the binaries, don't forget to add the *contrib* modules by adding a path to the *opencv_contrib/modules* folder. If you use the CMake GUI, you will be able to provide this path directly with the option *OPENCV_EXTRA_MODULES_PATH*.

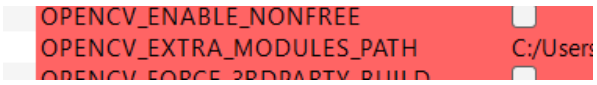

Figure 1: Adding extra modules in CMake GUI.

If you build the binaries from command-line, you will have to add this path manually with the option :

```
cmake -DOPENCV_EXTRA_MODULES_PATH=PathToOpenCVContribModules
```

Figure 2: Adding extra modules in command-line.

For wxWidgets, it is also recommanded (but not necessary) to build the binaries manually by downloading the Windows ZIP from the downloads page. You can use CMake again to build the binaries, and then proceed to the next step.

Finally, you will have to define multiple environment variables.

Note : If you do not create these variables, the makefile will use the `include` *and* `libs` *provided in the folder.*

For OpenCV, you will have to define the variable *OPENCVB_DIR*, and add the path to the `include` folder of OpenCV to this variable. This is the variable used for compiling the source files using OpenCV. You will also have to create the variable *OPENCVL_DIR* and add the path to the `bin` folder : we use this variable for the linking process.

For wxWidgets, you will again have to create 2 environment variables. Create the *WXB_DIR* and add the path to the `include` folder of wxWidgets. You may also have to add the *WXSETUP_DIR* variable and give it the path to the `setup.h` file, depending on how you installed the library. Most of the time, it will be located in the `lib` folder of wxWidgets or one of its subfolders. You will finally have to create the variable *WXL_DIR* and add the path to the `lib` folder of wxWidgets to the variable.

2 Linux distributions setup

On a Linux distribution, you will have to install the wxWidgets and OpenCV libraries.

You can follow this tutorial for setting up OpenCV. It is important in this tutorial to not forget to add the option *OPENCV_EXTRA_MODULES_PATH* with the correct path while configuring with CMake.
Note that if you install OpenCV without using `pkg-config` as shown in the tutorial, you may have to edit the makefile to compile the source files.

For wxWidgets, you can follow this tutorial. If nothing went wrong, you should now be able to compile the source files and create the executable using `make` in command-line.

3 Program structure

3.1 Folders organisation

The folder contains several subfolders, each containing specific files.

- **docs** : contains the user manual and the technical manual – this file.
- **headers** : contains all the headers files (*.hpp* files).
- **include** : contains the content of the `include` folders of OpenCV and wxWidgets.
- **installer** : contains the NSI script used for building the installer.
- **libs** : contains the libraries required for linking and running the program on Windows OS (*.dll* files).
- **obj** : contains the object files (*.o* files) created during compilation.
- **resource** : contains the software icon and the resource file (*.rc* file).
- **src** : contains the source files (*.cpp* files).

3.2 Source/header files organisation

Every source file (*.cpp* file) has its corresponding header file (*.hpp* file). This way, we have a lower compilation time, and we can re-use elements defined in a header file in any source file. This is also a way to prevent cross-referencing while using functions included from these header files.

One of the source files (`App.cpp`) does not have a header file, as its code is very short and is not subject to any big changes. Also, one of the header files (`structs.hpp`) does not match any source file : this header contains only the structures declarations, and some macroconstants.

In each header file matching a source file, there is a short description above the code explaining what this file contains. Finally, each source file matching a header file implements functions declared in this header.