Camera Manager

Technical guide

2019 version made by Hugo Fournier, Adrien Germain and Gauthier Leclercq
Based on the 2014 version made by Thomas Dubrulle and Antonin Durey
With Tomas Holt, Grethe Sandstrak, Alexander Holt and J.H. Nilsen

Table of contents

I. Set up the environment 5

1. Installation on Windows 5
1.1. Executables pack 5
1.2. FlyCapture and Spinnaker SDK 5
1.3 Microsoft Visual Studio 2015 6
1.4 Qt 7
1.5 Import of the project 7

2. Installation on Ubuntu 8
2.1. GCC/GPP Compiler 8
2.2. FlyCapture and Spinnaker SDK 8
2.3. Qt 9
2.4. Import of the project 9

II. Implementation 10

1. QtCreator and main 10

2. MainWindow 11
2.1. Main presentation 11
2.2. Camera Tree 11
2.3. Project Tree 11

3. Camera implementation 12
3.1. FlyCapture Implementation 12
3.2. FlyCameraManager 12
3.3. FlyCamera Parameters 12
3.4. Image capture 12
3.5. Trigger mode 13
3.6. Spinnaker Implementation 13
3.7. SpinCameraManager 13
3.8. SpinnakerCamera Parameters 13

4. Opening files 14
4.1. ConfigViewerWidget 14
4.2. ImageViewerWidget 14
4.3. CalibrationViewerWidget 15
4.4. SocketViewerWidget 16

5. WidgetGL 16

III. Miscellaneous 17

I. Set up the environment

1. Installation on Windows

The project has been written to use the defaults installation paths of all the software you are going to install. If you change the installation paths, you may have to modify the .pro file in Qt. You will get more details on that later.

1.1. Executables pack

In order to let the software work, you will need to install the FlyCapture and Spinnaker SDK, Qt and Microsoft Visual Studio 2015. Please note that the project won't work if you try to use another version of Microsoft Visual Studio, it has to be the 2015 version (but you can use community or professional version, it doesn't matter).

In order to make the downloads easier for you, we made an archive with all the executables files so that you don't have to download everything. Please note that if you use that archive, Qt will update and retrieve its files as you install it, Visual Studio is up to date, the FlyCapture SDK is up to date too but Spinnaker may have received some update since the release of the Camera Manager software (June 2019). Camera Manager is fully operational with the version included in the archive but you may need to update the SDK to implement other features later. Here is the link of the archive: https://mega.nz/#!rZc1UYLL!HJE8Mpwwa6zSB6-FQH6xYe8vH3EN_1Cnnz9Zqxg786Y

1.2. FlyCapture and Spinnaker SDK

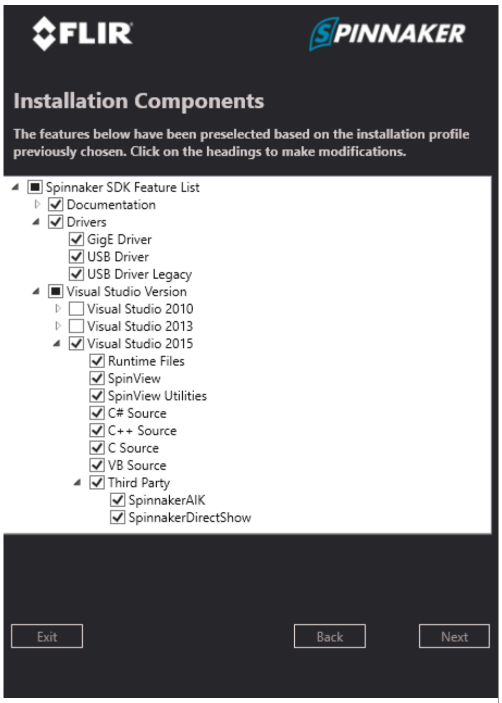
If you choose not to use the archive solution described above or if you need to download a more recent version of the SDK, just follow these steps:

1.2.1 FlyCapture installation

- Go to this link: https://www.flir.eu/products/flycapture-sdk/
- Click on the Download button and then Download from Box
- Click on the Windows folder then FlyCapture Latest Full SDK
- I recommend you to download both the 64 and the 32 bits version of the SDK

Now that you have the executables files, just launch the 32-bits installer, follow the steps, be sure to check the box "I will use USB cameras" and then "The installer will register the DirectShow DLLs".
When the first installation is over, repeat the process with the 64-bits installer.

1.2.2 Spinnaker installation



FLIR SPINNAKER

Installation Components

The features below have been preselected based on the installation profile previously chosen. Click on the headings to make modifications.

- ■ Spinnaker SDK Feature List
  - ✓ Documentation
  - ✓ Drivers
    - ✓ GigE Driver
    - ✓ USB Driver
    - ✓ USB Driver Legacy
  - ■ Visual Studio Version
    - ☐ Visual Studio 2010
    - ☐ Visual Studio 2013
    - ✓ Visual Studio 2015
      - ✓ Runtime Files
      - ✓ SpinView
      - ✓ SpinView Utilities
      - ✓ C# Source
      - ✓ C++ Source
      - ✓ C Source
      - ✓ VB Source
      - ✓ Third Party
        - ✓ SpinnakerAIK
        - ✓ SpinnakerDirectShow

Figure 1 : Components to install in Spinnaker installation

The download process of the Spinnaker SDK is roughly the same but the link is: https://www.flir.eu/products/spinnaker-sdk/

The installation process is slightly different. You have to check the box "Application Development" and then you have to make sure that the correct boxes are checked according to the Figure 1.

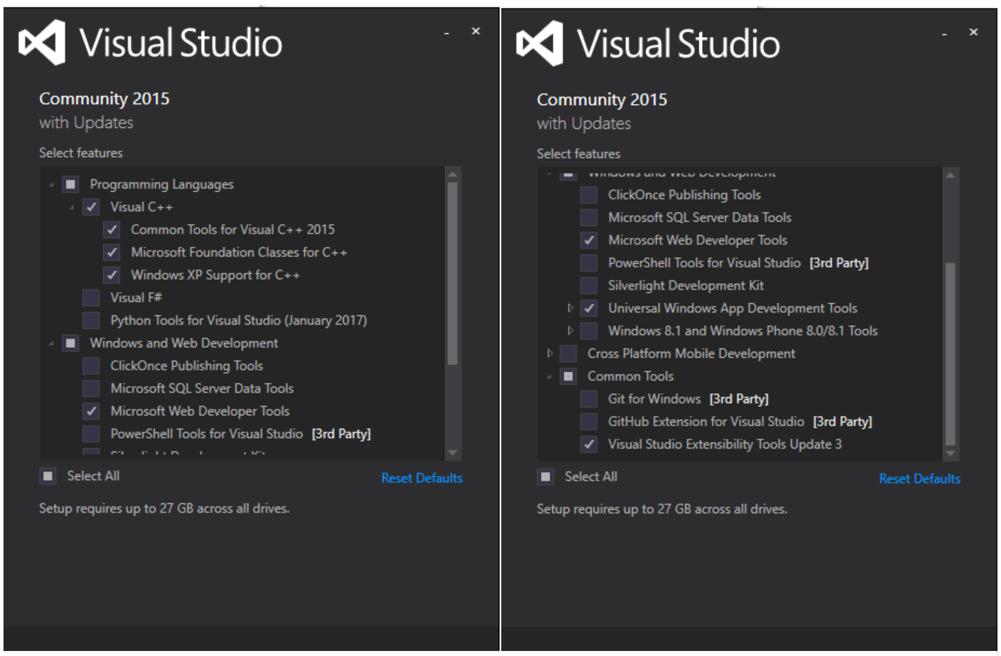
When the installation is over, please repeat the process with the other installer.

If you are asked to restart the computer, you can do it after the second installation.

1.3 Microsoft Visual Studio 2015

If you didn't choose the archive way, you have to find an installer of Microsoft Visual Studio 2015 which is pretty hard to do because of the fact that Microsoft is pushing the newer versions and make hard to find the 2015 version.

When you have the installer, run it and be sure to have all the components listed on the figure 2 checked in order to let the Camera Manager work properly.



Visual Studio

Community 2015
with Updates

Select features

- ■ Programming Languages
  - ✓ Visual C++
    - ✓ Common Tools for Visual C++ 2015
    - ✓ Microsoft Foundation Classes for C++
    - ✓ Windows XP Support for C++
  - ☐ Visual F#
  - ☐ Python Tools for Visual Studio (January 2017)
- ■ Windows and Web Development
  - ☐ ClickOnce Publishing Tools
  - ☐ Microsoft SQL Server Data Tools
  - ✓ Microsoft Web Developer Tools
  - ☐ PowerShell Tools for Visual Studio [3rd Party]

■ Select All — Reset Defaults

Setup requires up to 27 GB across all drives.

Visual Studio

Community 2015
with Updates

Select features

- ☐ ClickOnce Publishing Tools
- ☐ Microsoft SQL Server Data Tools
- ✓ Microsoft Web Developer Tools
- ☐ PowerShell Tools for Visual Studio [3rd Party]
- ☐ Silverlight Development Kit
- ✓ Universal Windows App Development Tools
- ☐ Windows 8.1 and Windows Phone 8.0/8.1 Tools
- ☐ Cross Platform Mobile Development
- ■ Common Tools
  - ☐ Git for Windows [3rd Party]
  - ☐ GitHub Extension for Visual Studio [3rd Party]
  - ✓ Visual Studio Extensibility Tools Update 3

■ Select All — Reset Defaults

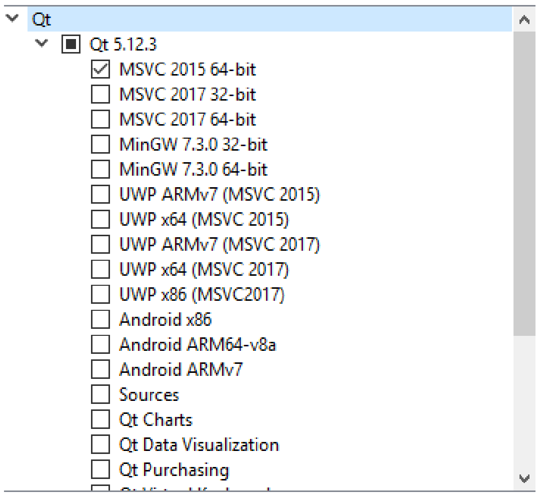
Setup requires up to 27 GB across all drives.

Figure 2 : Components to install with Visual Studio 2015

1.4 Qt

If you didn't choose the archive download, you can find the Qt installer by going to the Qt official website https://www.qt.io, clicking on the "Download, Try, Buy" button then on the "Go open source" button at the bottom of the page and then "Download".

Run the installer, you can skip the part where Qt asks you to create an account. At the component's selection screen, be sure to select the MSVC 2015 component as showed on the Figure 3 screenshot.



- Qt
  - [■] Qt 5.12.3
    - [✓] MSVC 2015 64-bit
    - [ ] MSVC 2017 32-bit
    - [ ] MSVC 2017 64-bit
    - [ ] MinGW 7.3.0 32-bit
    - [ ] MinGW 7.3.0 64-bit
    - [ ] UWP ARMv7 (MSVC 2015)
    - [ ] UWP x64 (MSVC 2015)
    - [ ] UWP ARMv7 (MSVC 2017)
    - [ ] UWP x64 (MSVC 2017)
    - [ ] UWP x86 (MSVC2017)
    - [ ] Android x86
    - [ ] Android ARM64-v8a
    - [ ] Android ARMv7
    - [ ] Sources
    - [ ] Qt Charts
    - [ ] Qt Data Visualization
    - [ ] Qt Purchasing

Figure 3 : Components to install in Qt installation

1.5 Import of the project

Keep in mind that we were working on the project while it was on the Desktop. The project dependencies contain relatives paths and works well when you let the default installation paths and when you have the project folder on the Desktop.

To import the project in Qt, open Qt and click on import project. Then, you have to go inside the camera manager folder and open the .pro file.
Then, you can click on the MSVC kit on the left-hand side of the screen to activate it.

Check the bar on the lower right-hand corner of the screen, if it becomes green, the first step is complete.
Then, you have to click on the hammer icon on the left-hand side of the screen to start the code compilation. Again, wait until the progress bar become green.

If you have some errors:
Click on the "edit" tab and open the project. Then, open the .pro file. You will now have to check if the different directories exist and if not, maybe change the path.
Start with the Qt directory. The relative path search for the Qt directory at the system disk root. Please note that the directory contains the version number and the project is made to search for the 5.12.3 version. If, in the Qt directory you find a directory with another version number, replace the path inside the .pro file with the correct version number.
If you still have a problem, check the installation paths of Spinnaker and FlyCapture; they have to match with those inside the .pro file and you have to change the .pro file if they don't.
If it isn't solved, check your installation and use the error codes and internet to find a solution.

If you have an error concerning the rc.exe file, you have to go into C:/Program Files (x86)/Windows Kits/10/bin/[highest version number]/x64 and then you can go to the .pro file to change the version number at the correct lines. You should go to the project tab and then click “details” in the “Build Environment” part. Then, look for the Path variable and add the path mentioned above at the end of the line then clean and try to qMake and build again.

When you have the green bar, it means that your compilation is done perfectly. Now, you can start the software by clicking the green “play” button.

2. Installation on Ubuntu

2.1. GCC/GPP Compiler

First, you need to install the GCC/GPP compiler. The installation is quite simple: open a terminal and run this command:

```
sudo apt-get install build-essential gcc
```

2.2. FlyCapture and Spinnaker SDK

- Go to this link: `https://www.flir.eu/products/flycapture-sdk/`
- Click on the Download button and then Download from Box
- Click on the Linux folder then download the SDK according to your system

The download of the Spinnaker is roughly the same but you have to go on this link instead: `https://www.flir.eu/products/spinnaker-sdk/`

Concerning FlyCapture, the “Xenial Xerus” version is for Ubuntu 16.04 and the “Bionic Beaver” one is for Ubuntu 18.04. Choosing amd64 file allows you to install it on Intel/AMD CPUs. If you are running on an ARM CPU, choose the “arm” file.

When you have downloaded and extracted the two files and, you can use the readme to continue the installation of the two SDK. You just have to copy-paste the commands that are in these files.

In case you are experiencing some troubles, with the installation of Spinnaker, you can try to raise the memory limit:
Open the /etc/default/grub file in any text editor.
Find and replace (You may open this file with the super admin rights):

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
```

With this:

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash usbcore.usbfs_memory_mb=1000"
```

Then update the grub with `sudo update-grub` and reboot the pc.

To check if it worked, execute this command:

```
cat /sys/module/usbcore/parameters/usbfs_memory_mb
```

If it returns 1000 it worked, if not, you can try to execute this command:
```
sudo sh -c 'echo 1000 > /sys/module/usbcore/parameters/usbfs_memory_mb'
```

Then, try again to install the Spinnaker SDK.

You can verify your installation by launching the FlyCapture software or SpinView.

2.3. Qt

You need to install Qt in order to launch and work on the camera manager project.
Go to the official Qt website `https://www.qt.io` then click on the "Download, try, buy" button on the upper right-hand corner of the page then click on "go open source" at the bottom of the page and click "Download".

When the file is downloaded, just run it and follow the instructions.
When it is done, launch QtCreator and click on Tools then Options (on the menu bar) then on the new window, go on the Kits tab and then Qt Versions. If you have a kit under the Auto-detected tab, you are done. If you don't have any kit Auto-detected, you have to add a new one back in the Kits tab. You have to set the right Qt version and set the compiler on GCC.

2.4. Import of the project

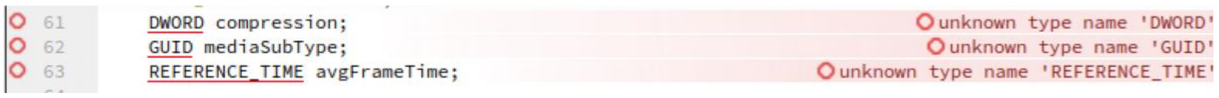
Now, you can click on File and then Import project and open the .pro file which is in the camera manager folder. Next, check the progress bar on the lower right-hand corner, when it is green, you can try to compile the code by clicking on the hammer button.
If the bar is green again, you can launch the project by clicking on the green "play" button.

If you have some issues:

in **imagedisplay.h**: you have to add:

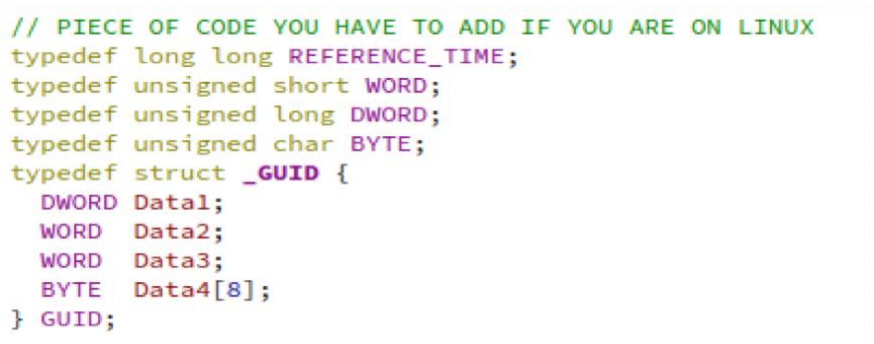
- `#include <math.h>` if sqrt function isn't working
- `#include <emmrintin.h>` if _m128i type isn't recognized (Linux OS)

in **PGRDirectShow.h** only if you are on Linux OS:



```
DWORD compression;               unknown type name 'DWORD'
GUID mediaSubType;               unknown type name 'GUID'
REFERENCE_TIME avgFrameTime;     unknown type name 'REFERENCE_TIME'
```

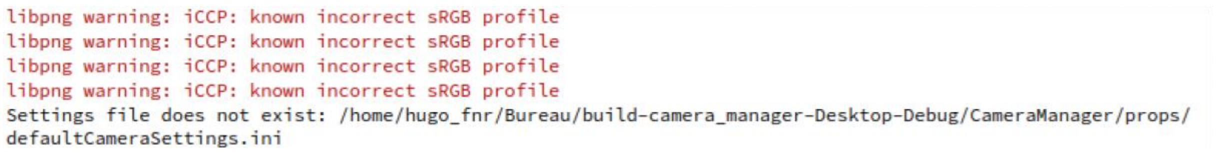
To solve this, you have to add this at the beginning of the header: in **the console when you build & run the program:**



```
// PIECE OF CODE YOU HAVE TO ADD IF YOU ARE ON LINUX
typedef long long REFERENCE_TIME;
typedef unsigned short WORD;
typedef unsigned long DWORD;
typedef unsigned char BYTE;
typedef struct _GUID {
  DWORD Data1;
  WORD  Data2;
  WORD  Data3;
  BYTE  Data4[8];
} GUID;
```

This is due to a missing folder in the build-camera_manager-Desktop-Debug.

The simple way to solve that is to copy the folder CameraManager which is inside the camera_manager main folder and paste it into the build folder.



```
libpng warning: iCCP: known incorrect sRGB profile
libpng warning: iCCP: known incorrect sRGB profile
libpng warning: iCCP: known incorrect sRGB profile
libpng warning: iCCP: known incorrect sRGB profile
Settings file does not exist: /home/hugo_fnr/Bureau/build-camera_manager-Desktop-Debug/CameraManager/props/
defaultCameraSettings.ini
```

Please note that you have to build the program at least one time to have the build-camera_manager-Desktop- Debug folder.

II. Implementation

1. QtCreator and main

If you succeeded all the configuration steps without troubles, then launch QtCreator. You can now load the project by going into file menu and import file/project. Then, select the .pro file which is inside the camera manager folder. It will load all the files (header, cpp and the ui files).

The *main.cpp* file is the entrance of the software. It will simply create a MainWindow.

2. MainWindow

2.1. Main presentation

The MainWindow class, is the main window of the software. It is divided into 4 parts:
- The menu bar: created with the MenuBar class
- The tool bar created with the *mainwindow.ui*
- The left menu: created with the *mainwindow.ui*, but considerably used in MainWindow.

It contains 2 tabs: camera and project (see next page for more details about that).
- The central widget: contains QMdiSubWindow classes (CalibrationViewerWidget, ConfigFileViewerWidget, ImageViewerWidget, SocketViewerWidget, and a QMdiSubWindow in the activeCameraEntry class; which is an internal class of AbstractCameraManager, see next page for more details about that).
The *mainwindow.h* also contains 2 internal classes, which are 2 *QThreads*. They are classes to detect automatically new cameras, and update automatically the camera properties.

In the MainWindow class, the code is cut in 6 parts:
- the main functions : constructor, destructor
- the toolbar functions
- the menubar function : there is only a slot
- the camera tree functions : right click, double click...
- the thread functions : there are 2 functions, because there are 2 internal threads - the project tree functions : right click, double click...

2.2. Camera Tree

The cameras you plug will be automatically detected thanks to the ThreadDetectCamera class, which is an internal class of MainWindow (see *mainwindow.h*).

In the MainWindow, you will find functions for the right click, the 4 actions coming from the right click (about group and name) and functions about changing the current item.

2.3. Project Tree

In MainWindow, you will find functions for the right click, loading projects (which is done recursively with createTreeFolder and createTreeItem), and double click which is the way to open files.

3. Camera implementation

3.1. FlyCapture Implementation

The management of these cameras is divided into two class: FlyCamera and FlyCameraManager. These classes extend AbtractCamera and AbstractCameraManager to implement the FlyCapture API into the Camera Manager software.

For more details on the implementation of each class and its methods, please refer to the Doxygen documentation.

3.2. FlyCameraManager

The FlyCameraManager class implements the AbstractCameraManager and contains a vector of FlyCameras.

The detectNewCameras method detect all connected cameras thanks to the BusManager which contains a list of all physical cameras connected. The BusManager is included in the FlyCapture API.

3.3. FlyCamera Parameters

Management of parameters is simple and is based on two methods: one to set value to the camera, and another to get a value from camera in order to update the GUI. In case of FlyCapture cameras, we have a pointer to the current camera instance. Each property has an automatic mode and a manual mode which allows you to set value you want with the slider. The chosen mode and value are updated at the same time. The value of a FlyCamera property can be an integer or a float number, so a single FlyCamera property has three attributes to update with the same value in order to be sure to always set all types. When there is an update, the value is fetched from camera, and to decide which value is the correct one, the method tests the number of decimals (set in CameraManager) and choose valueA for 0 decimals or absValue for 1 or more.

3.4. Image capture

First of all, we need to set the camera in capture mode with the StarCapture method, then we need to get the image with retrieveBuffer method. Once the image is stored, we must convert it in order to be able to display it in the Qt interface. As the image from FlyCapture cameras is a 8bits and we need a 32bits image, we reproduce the bits as to obtain a 32bits black and white image. When live view mode is on, startAutoCapture method is called in FlyCamera class, which will loop and grab an image until live view is turned off, that is to say when stopAutoCapture method is called.

3.5. Trigger mode

In the GUI, the trigger mode of the cameras can be set on and off via the camera properties block. This setting allows the user to use an external trigger to control the image capturing.

The trigger mode has 4 parameters: onOff, mode, parameter and source. To configure an external trigger, we have to set mode, parameter, source to 0, and onOff to true. When the automatic trigger is off, hitting the capture-one-image button would freeze the program because it's waiting for the external trigger. The software trigger is therefore fired in these conditions to avoid that problem.

3.6. Spinnaker Implementation

The management of these cameras is divided into two classes: SpinCamera and SpinCameraManager. This classes extends also AbstractCamera and AbstractCameraManager to implement the Spinnaker API into the app.
The methods inside both of the classes are the same as FlyCamera and FlyCameraManager, they are just adapted for Spinnaker.

3.7. SpinCameraManager

The *SpinCameraManager* class implements the AbstractCameraManager and contains a vector of SpinCameras. It defines the list of properties which are available with the Spinnaker cameras and that will be used in the SpinCamera object. Each property is set manually and contains a name that refer to the camera property. Each property contains also minimum and maximum values which are mostly used for the GUI since the Spinnaker Camera automatically adapt the value if it goes out of range.
For SpinCameraManager, there is no detectNewCameras, this is directly implemented into the detectNewCamerasAndExpand method in AbstractCameraManager. To ensure the cameras are still detected, we use the SystemManager class we created with the cameras list. The camera list is getting from the System class in Spinnaker API.

3.8. SpinnakerCamera Parameters

Parameters management is simple and is based on two methods: one to set value to the camera, and another to get a value from camera in order to update the GUI. In case of Spinnaker cameras, we have a pointer to a reference of the current camera instance. Each property has an automatic mode and a value which is set and update at the same time as the value. The value of a *Spinnaker* property can be an integer or a float number, so a single Spinnaker property has three attributes to update with the same value in order to be sure to always set all types. When there is an update, the value is fetched from camera, and to decide which value is the correct one, the method tests the number of decimals (set in CameraManager) and choose valueA for 0 decimals or absValue for 1 or more.

4. Opening files

4.1. ConfigViewerWidget

The structure of the option file wizard is composed of two classes, in an effort to keep the flexibility at its maximum.

The first class, called ConfigFileViewerWidget, show the window in the main area of the application. Then, it reads the information contained in the option file with the second class, the “ConfigFileReader”, so that it displays them on the screen. The advantage with this structure is that if the format of the file is not significantly changed, the ConfigFileReader class should not need to be changed at all, as it performs all tasks that the ConfigFileViewerWidget needs: reading one parameter with type checking or not (in the case we don't know its type) at the desired position.

On its side, the ConfigFileViewerWidget has been cut into several methods that add each, one part of the window. It has been designed so that it is easy to modify the way the wizard work.

There are several methods that add a specific widget to edit a certain type of parameters. They all follow the same pattern: they create the widget and add it to the wizard layout with the specified parameter at the given position. They all returns the newly created widgets so that they can be used elsewhere.

If you want to add another type of parameter, only the createWizard and saveWizard method need to be changed by adding a new condition in it. You may also add a new method to add the new widget to the wizard.

To save the wizard into the QtextField, the saveWizard method use lists of parameter indexes along with their corresponding widget. It then loops on all of them, and replace the old value with the one contained in the widget.

You can notice there are two classes in the ConfigFileViewerWidget: the ConfigFileViewerWidget itself and the pathEditBox. The PathEditBox is simply a field with a button that allows the edition of paths with an explorer, which is more convenient than writing it yourself. If you want, you can move it to another file and use it for other plug-ins, and add an include in the configFileViewerWidget, since the two aren't normally intertwined together.

4.2. ImageViewerWidget

The ImageViewerWidget class is responsible of opening the **grupper images**. There are 5 functions inside:

- the constructor
- initializingImage: initialize the image with the time number provided in parameter

- initializingPoints: initialize the points according to the number time (kept in mind as attribute in the class)
- mousePressEvent: click on the image
- wheelEvent: using the mouse wheel.

4.3. CalibrationViewerWidget

The CalibrationViewerWidget class is the class to open and read the **calibration_summary** file. We can cut the functions in 4 parts:

- the constructor and initializing functions
- the changing view functions: showTextView and showTableView
- the right click function and assimilated
- the left click function and assimilated

The showTextView function put a TextEdit as main widget. This widget is in fact a CalibrationEdit, which is an internal class of CalibrationViewerWidget. It has been reimplemented to be able to use mousePressEvent.

The left click functions are only used in the text view. In its assimilated functions, you will find functions as select, calculateShowFailed, calculateShowUseless, calculUselessCombinations, and the two functions used to make the combo sort: sortCombo and executeSortChange.

These functions are sometimes really difficult to understand. In the computing functions, keep in mind that the enum type Calibration is crucial to know which in which state is each combination. In the functions which display lines, or not, the best thing is the function moveCursor(QTextCursor::MoveOperation, QTextCursor::MoveMode) to select some text, removeSelectedText() to remove text, and insertText() to insert text.

If you are encountering some troubles with these functions and you want to remove them, pay attention to the consequences that it will draw because there are some loops that can mess with the project. You should consider removing the loops if you need to remove these functions.

Many comment lines have been put to try to help you at best to understand how we thought it and how we made it.

4.4. SocketViewerWidget

The SocketViewerWidget class is the class to open and read the **socket** file, which is the file which contains the 3D data. We can cut the method in 4 parts:

- the constructor and initializing functions
- the changing view functions: showTextView, showTableView and show3DView
- the right click function and assimilated
- the assimilated functions for the table view: areaBarsMoved, displayToolTip, valueChanged and getTimeSlider

There are only assimilated functions for the table view, because the text view does nothing, and the functions for the 3D view are in the WidgetGL class (see next page).

The widgets containing the values are CoordinatesLabel, which is an internal class extending QLabel. It has been reimplemented to be able to use the mouseMoveEvent function, which allow the QToolTip to be displayed.

5. WidgetGL

The WidgetGL class is the class which inherits from QGLWidget, which is the **QtGL** component to use **QtGl** and **OpenGL** functions inside. There are 8 functions in this class:

- the constructor
- initializeGL: initializing function coming from QGLWidget
- initializingCameraCoordinates: function to initialize the cameras coordinates.
- paintGL: which is the function where you put all your painting desired.
- resizeGL: function to resize cleanly 3D painting area according to the widget size
- showView: show the view according to the time provided as parameter
- eventFiltrer: check all the events (right and left click, key pressed and released and wheel event).
- setXRotation(), setYRotation(), setZRotation() : setting the new rotation angle.

This class may also contain code which is not working. This is because we have not enough time to finish it:

- the right click: it should provide default view angle, so that the user would not have to deal with X, Y, Z key, and wheel mouse
- the camera points are well initialized, but not drawn every time. Seem to have a problem, maybe about depth...
- the left click: clicking on a point to have its number and coordinate.

Currently, the conversion from 3D coordinates to 2D is not working.

III. Miscellaneous

Tips:

- **You may find in the files several comments named "//TTODO"**. They tell improvements or bug fixes we didn't have the time to do. Usually they have some explanations in them, as well as some ideas on how to do them. I hope they will be useful if you want to improve even more the application.
- **You can find most, if not all, of the methods you need in Qt**, which have similarities with Java standard library. Using them instead of system dependent methods ensure that you keep your project multi-platform.
- **The installation of the project (not the software themselves) on Windows is more complex than on Linux**, but it may be useful to have someone working on Windows while the other is on a Unix system to constantly check the compatibility of the project on these two platforms.
- **On Visual Studio, if you get an LNK error (generally at the beginning when you install the project), it is often because the linker doesn't find a method, a class or a library in all the files you have included**. Check in the .pro and in the .pri files that the paths are all good.