Condor Suite Imaging System
Development Guide

Authors:
Marc-Antoine Lalonde
Vincent Girouard

Reviewers:
Marc-André Tétrault
Amir Sajjadi

1 Table of Contents

2 Introduction

The goal of this document is to help a software developer to understand the architecture of the Condor Suite software and start developing as fast as possible. This document includes a summary of the necessary technologies, as well as several UML diagrams describing the relationships between the different classes of the project.

3 Naming and structure

The project is referenced here as Condor Suite, unlike the user guide where it is referenced as Condor Viewer. In fact, Condor Viewer is only a portion of Condor Suite, but it is the only part that is accessible to the user. At the time of writing, Condor Suite is also composed of CameraDrivers, ImageProcessingToolbox, LaserController, and VoiceRecognition. As a software developer, you will be working with Condor Suite, which is a Visual Studio 2015 C++/Qt project.

In this guide, several diagrams will be presented. These diagrams were made using draw.io, a free website. In the diagram folder can be found all diagrams presented in this guide, as well as files with no extension that can be loaded on draw.io for editing.

4 Technologies

Condor Suite is composed of several technologies, listed below:

Visual Studio 2015: the IDE used to develop the project.
Qt 5.9.4: a large open-source LGPLv3 C++ library use throughout the code, on which the GUI is based.
Qt VS Addin: the Visual Studio Plugin that allows Visual Studio 2015 to build Qt projects.
OpenCV: a large, open-source image processing C++ library used greatly for managing images throughout the project.
Matrox Imaging Library (MIL): The SDK used to control the Blue Vision camera in the Fiat-L system. It is only used in the CameraDrivers project.
XCLIB: The SDK used to control the cameras used with an Epix frame grabber. Currently the Condor3 and the Ninox 640.
CMUSphinx: This is an open source library used for the voice recognition.
Exiv2: This is a library used to manipulate and save Xmp metadata in the image files.
Gitlab.com: a versioning website that supports integration with various git software, such as Git Kraken and SourceTree. Use of this service requires a gitlab.com account. Access to the project can be requested from Marc-André Tétrault at MTETRAULT@mgh.harvard.edu.

Notes: VS 2015 was the latest version that had a compatible Qt plugin when the project was first started. The software is known to work with that version, but may possibly be upgraded in the future.

5 Setup a developing environment

1. Install Visual Studio 2015 on the target computer. During the installation, include the C++ language as well as the Windows 10 SDK.
2. Install Qt 5.9.4 on the target computer
3. Download and install the Qt VS addin for Visual Studio
4. Install your favorite Git software and clone the Condor Suite project to your computer.
5. To run the program using a camera running with the MIL, you will need the MIL correctly installed and a valid license. Contact rlaxogns33@nawoovision.com for any support.

You may now open the project in Visual Studio, configure the Qt addin so that it links to the correct installed Qt version as shown in the figure below and compile the project. You may also have to right-click on each project in the solution explorer, select *Qt Project Settings* and configure the project to use the Qt version you just configured.

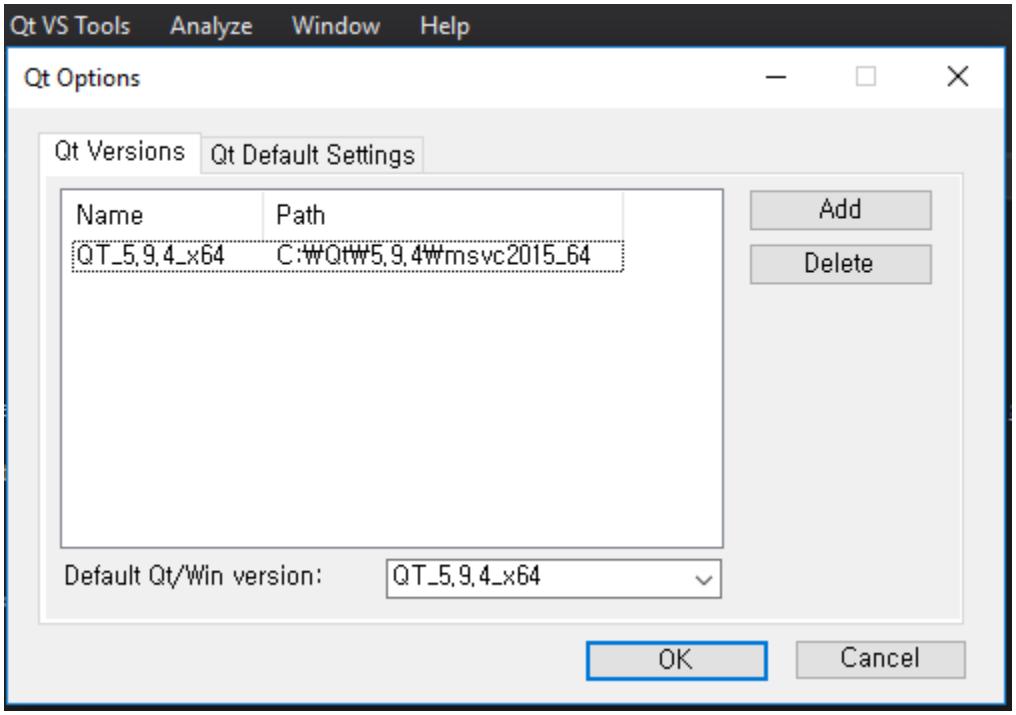

Figure 1: The top menu bar in Visual Studio 2015 and the Qt options dialog correctly configured

Note : The QT addin generates a vcxproj.user project file with the QTDIR environment variable, local to the VS project. This file is not and should not be included in the GIT repository. However, on occasion, the QT addin does not always regenerate this file, which causes compilation to fail. An alternative method to add the missing reference is to register the variable directly in the windows environment. In windows explorer, right-click on “computer”, choose “advanced system settings”, “environment variables, and add your local installation path to the profile or system environment. For example, see figure 2.

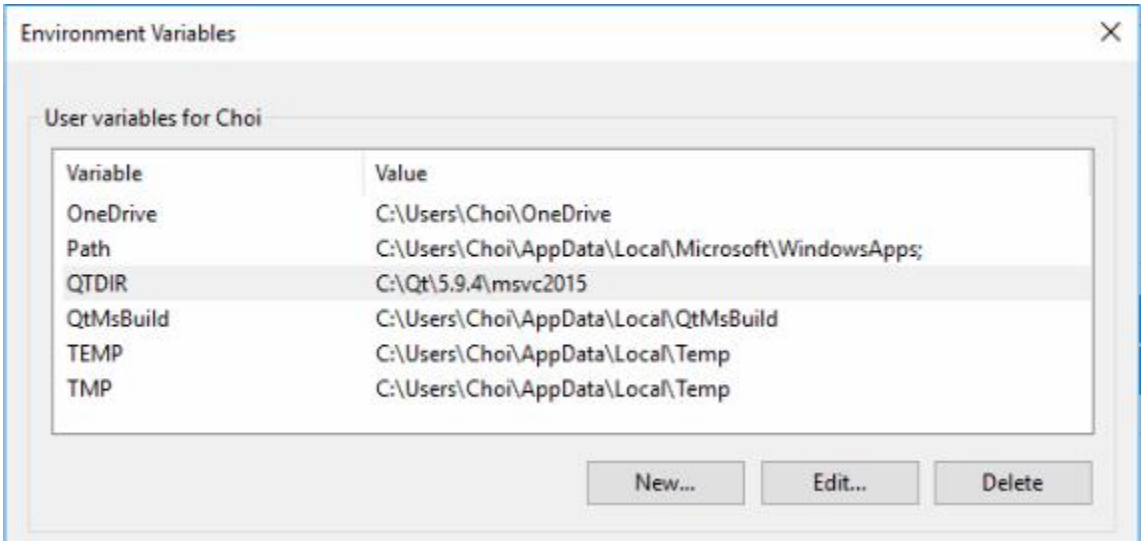

Figure 2 QT path environment macro in the windows settings.

6 Program architecture and management workflow

The architecture chosen here is based mainly on the MVC pattern / group of patterns, and incorporates some elements from the MVVM pattern. Note that the Qt version of the MVC pattern merges the Controller and View, which may be confusing to those who are not used to this architecture. Also note that design patterns are not rigid structures, but rather flexible principles that should be adapted to each situation. One should make sure that he/she understands the previously mentioned concepts before modifying the software. To get you started, here are interesting reads on design patterns and software architectures:

Wikipedia – MVC pattern
Qt – MVC pattern
Wikipedia – MVVM pattern
Another great programming resource is Stack Overflow, which you should already know about 😉.

The issue tracking for new features and bug correction is all done through GitLab. All discussion regarding implementation details, user preferences and etc. should be done in the comment section of each issues, avoiding emails as much as possible. This will help keep a good documentation of implementation and design choices.

7 Class and image flow diagram

The Figure 3 identifies the main classes of the Condor Suite project and the relationships between them.

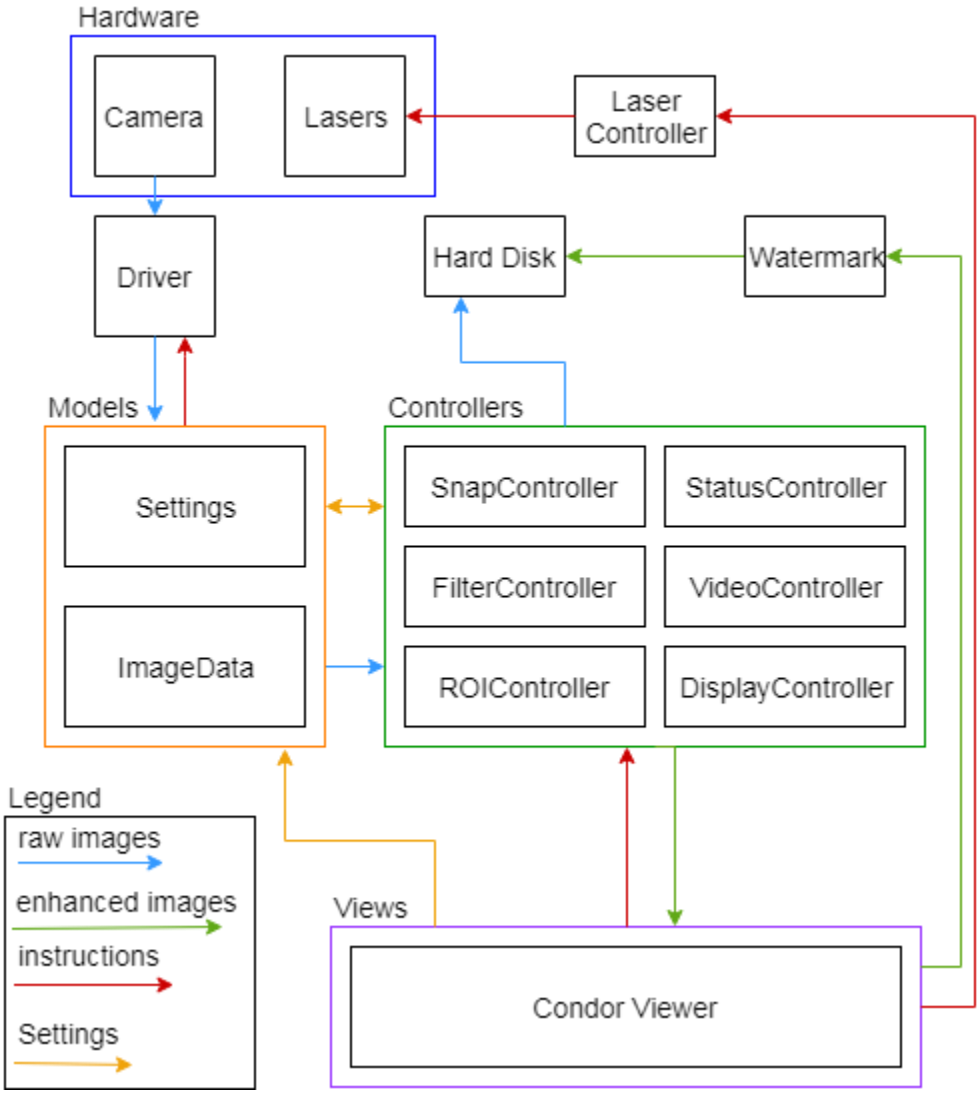

Figure 3: Condor Suite Class diagram and image flow

Additional classes can be found in the code that are not represented on this diagram for clarity purposes. Those classes are essentially widget classes or support classes. They are, for the most part, specialized subsections of the view. For example, the class HistogramWidget, which is not represented on this diagram, is responsible for showing the histogram as one may have guessed, and is owned by the Viewer (main window).

The Figure 4 is a simpler view presenting how the images themselves flow from class to class and what is done to the images in each class.

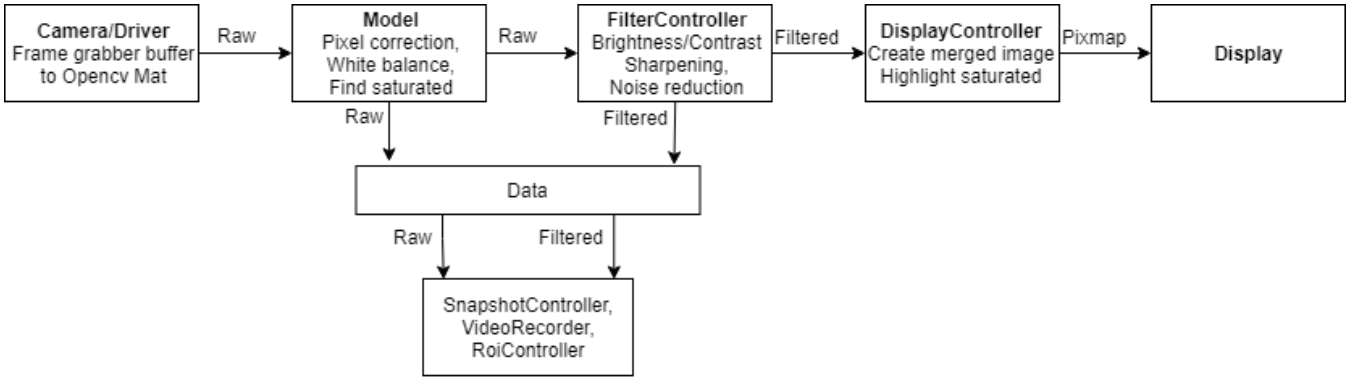

Figure 4: Condor Suite Images flow diagram

The diagram shows the flow of the images from the camera to the display:

1. The images start in the camera and driver as a buffer on the frame grabber. These images are converted to Opencv Mat objects in the driver.
2. The Model (This class is badly named, it acted as a "Super do-it-all" class at first. Effort as been made to separate everything in more specialized classes. Now the only things remaining are transformations and operation to apply to the raw images. It should be renamed eventually) is then responsible for doing the defective pixel correction, the white balance and finding the saturated pixels. It also applies the transforms (rotation and flip). The raw image is then sent to the FilterController and the Data object, which will keep an exact copy.
3. The FilterController will do the brightness and contrast correction and apply the various filters. It then sends the filtered images to the DisplayController and the Data, which will keep a copy, independent of the raw images.
4. The DisplayController, which is in fact one controller per display, does different things depending on the channel. For the NIR channels, it will apply the saturation overlay. For the merged channel, it will merge the channel and apply the correct colors to the NIR images. It will then send a pixmap to the display.
5. The Display will then simply draw the pixmap for visualization.

On the lower part of the diagram is a block containing the other controllers. It simply shows that these controllers are not in the normal 'flow' of the images. They will instead fetch the images from the Data object when required as they do not do a continuous task. Depending on the controller, they will fetch the raw image, the filtered image, or both.

8 Maintenance, non-user configuration and setup tools

8.1 Generate defective pixels

* See appendix A for a detailed documentation on defective pixels

The lists of defective pixels for the NIR1 and NIR2 channels can be updated at any time through the Hardware Settings dialog.

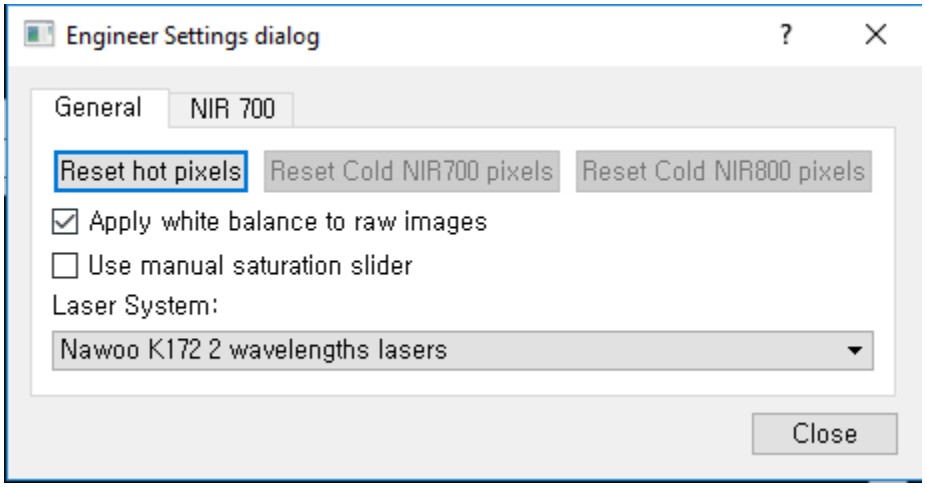

Figure 5: Resetting defective pixels

As one can observe on the Figure 5, the list of hot pixels can be updated simultaneously for both the NIR1 and NIR2 channels, as it only requires closing the lens' shutter. However, updating the cold pixels must be done separately for each channel, as they each must be flooded by a different wavelength.

This procedure should be done when first setting up a new computer, or when swapping cameras on a system, as no two cameras have the same defective pixels. Refer to the Condor Suite Installation Guide for a detailed procedure.

*** This procedure should be done by users that know what they are doing *and why*. This procedure should not be repeated more than every few months, because hardware malfunction is not 'normal' and should not vary a lot over time.**

8.2 Adding support for a new camera

Adding a new camera in the software can be done without changing a lot of existing code. The first thing to do is to create a new class for the driver. This class should imperatively inherit from the ICameraDriver interface and overload the pure virtual methods.

Refer to the existing drivers as examples, but here are the main things to consider:

1. The driver should expect to receive gain and exposure updated through the SetProperty method. Gain is in dB and exposure in milliseconds, the driver should do the necessary conversion.
2. To send the images, the driver should use "emit newImage(QVector<cv::Mat>)", where the vector contains 3 images in the following order: Color, NIR1 and NIR2. For fewer channel camera, black images of the correct size should be used to fill the vector.
3. The driver's constructor should do a lot except initializing member variables. The camera connection and configuration should be done in the "Init()" method, where a "noCameraFound" signal should be emitted if connection or configuration fails.
4. The software needs to receive a camera profile of the connection succeed. The end of the Init function is a good place to call "emit notifyCameraProfile(profile);".

To make the new camera appear in the dropdown selection menu, it is required to add the camera to the camera enumeration and the string map, located in the ICameraDriver header file. Then, add the corresponding entry in the 'switch' statement in the main function in order to instantiate the correct camera implementation.

8.3 Adding more metadata fields

The metadata uses the Xmp metadata standard, using the Exiv2 library. Example code can be found here. In Condor Suite, the metadata is mainly managed with the "CustomMetadata" class. The class has methods to easily serialize and deserialize to and from the XmpData objects and format the metadata to a string or html array.

The metadata feature was made with flexibility in mind. Thus, only the namespace must be registered in advance and not the specific fields. It is recommended to use the same namespace that is currently used for a sake of compatibility with the ImageJ visualization plugin. See **Appendix E** – XMP Metadata tools for more details on this plugin.

When adding a new generic field:

1. A new member should be added to the CustomMetadata class. This member should be of the correct type to hold the field's value.
2. The field should be added in the Serialize and Deserialize methods, following the same syntax as the other fields.

When adding a camera-specific field:

1. An XmpData object should be created and updated in the camera driver.
2. The object should be sent to the software with the "newMetadata" signal when the fields are update. See the Ninox640Driver for an example on how to achieve that.

8.4 Adding support for new lasers

The laser systems are more complex than the cameras and require more components. Namely, they require:

- A controller class, inheriting from ILaserController, responsible for the laser specific logic and serial communication.
- A laser widget, responsible for the user inputs that will act on the lasers (generally the generic widget is enough and doesn't need to be reimplemented).
- An advanced laser widget, inheriting from IAdvancedLaserWidget, responsible for the control tab in the 'Hardware Settings' which is mainly used to send a new configuration file to the lasers. A single widget can be shared for all wavelength (like the K172) or be independent (like the Arroyo). It depends on the laser systems. For the K172, they are shared because they share the COM port for all wavelengths.

To instantiate and connect everything correctly, one should add the laser type to the enumeration and string map in the LaserSystemsFactory class header. Then, a case statement should be added to LaserSystemsFactory::Init method. Look at the other laser types for examples.

8.5 Notes on the working distance sensor

- The distance sensor is made of an Arduino Nano and a HC-SR04 ultrasonic sensor. The Arduino sketch can be found in the Tools subdirectory of the repository.
- In theory, any type of Arduino can be used, but when dynamically finding the COM port in Condor Viewer, we look for a specific friendly name that might be unique to the Arduino Nano.
- The sketch was also made for a specific mount on the laser heads assembly (ring) in South Korea where the head of the sensor would sit 4mm bellow the lasers, therefore we add 0.4 to the distance value in the sketch. This should be modified when installing a sensor on a new system.

9 Updating the documentation

The documentation should be update at least with every release to reflect the changes. The documents to update are the user guide, the development guide and/or the installation guide depending on the changes.
When updating these documents, the review feature of MS word should be used to track the changes. The documents should then be sent via email to a reviewer. Some iterations can be done via email until all the changes have been accepted. Once the changes have all been accepted, the documents may be committed to git.

10 Software release versions and generation procedure

The software release involves several steps, necessary to ensure proper communication, bug tracking and bug fixing between the development team and the end users. First, the versioning nomenclature uses year/month format. Second, versions untested by the users are marked as “Release Candidate”, or RC. For example, “Condor Viewer 2018.04.RC1”. Once the RC has run for a while with the users, the program is promoted to a full release, such as “Condor Viewer 2018.04”. Bug fixes to full releases increments a counter added at the end, i.e. “Condor Viewer 2018.04.1”.

When creating an official release, the developers must follow this list:

1- Update the change log in the x64 directory, if not done. Do not mention the release number yet.
2- Create a release branch in Git, with the intended number
3- Modify the version number in the change log file
4- Modify the footer in the word documentation to reflect the new release
5- Commit these changes also to the pdf files in the x64/doc directory
6- Change the version in the help menu (hardcoded for now)
7- From a clean “git clone” of the branch, compile the .dlls and .exe. This validates no ignored files modify the compiler’s behavior.
8- Commit these the .dlls and .exe to git
9- Create the tag, and delete the local working copy.
10- Clone the tag
11- Package the x64 directory to a zip file
12- Upload to the “MIS Software/CondorViewerInstallers” folder in the Choi Lab’s Dropbox, naming the archive “CondorViewer_20XX.XX.zip”, adding extra characters as needed (RCX, for example).

When moving from RC to full release, follow the similar procedure, but start from the branch instead of the master.

Appendix A – Defective pixels documentation

Definitions

Defective pixel: Pixel that is either hot or cold due to hardware malfunction.
Hot (Live) pixel: Pixel that looks like it has some signal in it, even if the image is dark.
Cold (Dead) pixel: Pixel that looks like it doesn't receive any signal, even if the image is bright.

The issue

Defective pixels are somewhat of an issue, because they can impact different aspects of an experiment. Firstly, these pixels can be very distractive to the user. Even if they are small, their big contrast compared to the rest of the image makes them stick out more and catch the eyes a lot. Secondly, they slightly affect the quantification by raising or lowering the perceived value of the image. They achieve that by modifying real values in the data, which then affects the images statistics. Although it is shown that their impact is not very significant on quantification (see Validation scenario below), removing the pixels will still be beneficial if the method used does not compromise the good pixels.

Proposed solution

Fixing those pixels has been determined to be important to avoid distracting the user and mitigate quantitative impact. The proposed solution is a two steps process, to identify and filter out those pixels, which is simple and minimize disturbances by removing distractions and lowering the impact on quantification, as explained in this appendix. Three identification methods were explored for the identification step, detailed below, while the median filter was chosen for the correction method. This correction method has shown good results and is vastly used in the imaging industry. They will all be presented in the following lines to help keep a history of those iterations and justify the implementation choices.

Identifying the defective pixels

***This is not a detailed SOP, but more a high level explanation of the idea behind the procedure, the official SOP should be read and understood before attempting to modify the defective pixels lists.**

Setup

Hot pixels

To identify the hot pixels, one must produce an image in the darkest lighting conditions as possible, with the least noise and outside perturbation as possible. To do that, the user needs to close all the lights in the room, close the lens aperture totally and then take an image using a long exposure time (around 2000ms). This will produce a dark image, while the long exposure gives time to the hot pixels to build up a significant value.

Figure 6 Close-up of a dark image produced following the described method. The red circle show hot pixels that are clearly visible.

Cold pixels

To identify the cold pixels, one must produce the brightest and most uniform image possible. To do that, the field of view needs to be flooded be a light source of the correct wavelength for the channel. Then, capture an image which will turn out to be very bright, except for the dead pixels. At the time of writing this document, the lab does not have access to a proper uniform and controlled light source to safely and correctly flood the camera sensor. Thus, the cold pixel identification is not yet supported. When a proper light source is found and tested, different exposures, gain and threshold combination will have to be attempted to find the identification method that suits the need of the lab.

Threshold selection

a) Description

When deciding which pixel is considered to be defective, a threshold is applied to the image. All the pixels above the hot pixels threshold will be considered hot and those bellow the cold pixels threshold will be considered to be cold. The efficiency of the methods then comes down the threshold selection. To achieve good result, the threshold must be such that it finds as many defective pixels as possible, without interfering with the healthy ones.

b) Methods

i. Absolute 0.01% threshold

The first method that was tested is an absolute threshold. All pixels in the 0.01% brightest/darkest pixels were considered defective. The main advantage of this method is that it is simple to understand and implement. On the other hand, this method will depend on the gain value, because the overall brightness of the image is affected by the gain while the threshold will remain the same.

ii. Absolute 100 threshold

The second method that was used was to treat the 100 brightest/darkest pixels as defective. It is very simple and fast to implement and execute. Unfortunately, this threshold might be too low for older cameras or too high for newer cameras with less defects, removing too few or too many defective pixels.

iii. Relative 5 sigmas threshold

The third method uses the mean value and the standard deviation (sigma) as follow: The threshold is considered to be exactly five sigmas of the mean value (above or below). Every pixel that is higher or below the threshold is considered to be hot or cold. This method is much more robust as it takes into account the sensor's gain. The main drawback is that it is slower as it requires computing the average and standard deviation to identify the pixels.

c) Comparative results

The three methods were tested using 6 different images captured with different settings using the same camera to make sure to have a valid comparison. Here are the results:

Sample	Exposure	Gain	0.01%(i)	100 First(ii)	5 Sigmas(iii)
#1	**200**	**1**	0	100	56
#2	**2000**	**1**	0	100	94
#3	**200**	**18**	1	100	61
#4	**2000**	**18**	12	100	331
#5	**200**	**36**	11	100	100
#6	**2000**	**36**	83	100	754

Table 1 Number of hot pixels found for each identification method, for six different sample images.

Sample	Exposure	Gain	0.01%(i)	100 First(ii)	5 Sigmas(iii)
#1	**200**	**1**	4055	137	139
#2	**2000**	**1**	4055	152	159
#3	**200**	**18**	4055	217	234
#4	**2000**	**18**	4055	334	279
#5	**200**	**36**	4055	894	904
#6	**2000**	**36**	4055	1777	1025

Table 2 Equivalent absolute threshold values. All pixels above these values were considered to be hot. For the sample images, the bit depth was 12 bits, which means that the maximum value for a pixel is 4095.

Sample	Exposure	Gain	0.01%(i)	100 First(ii)	5 Sigmas(iii)
#1	**200**	**1**	0.000%	0.011%	0.006%
#2	**2000**	**1**	0.000%	0.011%	0.010%
#3	**200**	**18**	0.000%	0.011%	0.007%
#4	**2000**	**18**	0.001%	0.011%	0.036%
#5	**200**	**36**	0.001%	0.011%	0.011%
#6	**2000**	**36**	0.009%	0.011%	0.082%

Table 3 Percentage of the total number of pixels that were found to be hot for each identification method, for six different sample images.

This data tells us that there is no perfect solution for identifying the hot pixels and that each of them will heavily depend on exposure and gain in different ways. It has been determined from one of our contact with the industry that a defect rate of about 0.01% was an acceptable margin for the present camera. Having that in mind, it is possible to compare the methods and choose one.

i) To get to the 0.01% defect, the method of the 0.01% brightest required a high gain (sample #6, 36dB) which amplify the image noise. Amplifying such noise before doing any processing with the image is generally bad and not desirable.
ii) The method of the 100 first pixels is producing a much more consistent result selecting 0.011% of the pixels every time. Unfortunately, as mentioned earlier, the method does not adapt for better or worse camera in terms of defective pixels where the actual defect rate would be higher or below 0.01%.
iii) The 5 sigmas method is detecting a defect rate of 0.01% at an exposure of two seconds and a minimal gain (sample #2), which is near enough our estimated target. This result is promising, because it can detect a good amount of defective pixels without inducing noise. Also, it is less consistent than the 100-first method, but the lack of consistency is not as bad as it may look at first sight. It is shown that for a very high gain (sample #6, 36dB of gain), the rate of defects gets to 754 pixels, or 0.082% of the total amount of pixels. Correcting these pixels with the median filter method (explained in next section) will likely not affect the quantification data significantly, as shown in the validation scenario below, as these pixels are generally not clustered together.

The 5 sigmas method was selected and implemented in the software because of its relative threshold which is expected to adapt well for different cameras. Also, in controlling the exposure and gain for the identification, the defect rate can be very close to the expected value of 0.01%. In the end, even if too many pixels were identify as defective, correcting them with a median filter of a radius of one is not invasive on the image and will still produce accurate quantification values.

Correcting the hot/dead pixels

Once identified, the hot and cold pixels are corrected using a local median filter. No other solutions have been tested, however the median filter is the industry standard and provides very good results (see Example below).

Median filter

This method can be used on hot or cold pixels, achieving the same result as it does not require any prior knowledge of the pixel value, only its location. The idea is to look at the eight direct neighbors of the defective pixel and set the pixel to the median value of all the neighbors. The new pixel value will blend in with its neighbors, making it less distracting and practically impossible to distinguish for the user. Also, by using the median value, the corrected pixel will likely be very close to the value had it not been defective, making it a simple non-invasive method. Impact on quantification will be demonstrated below.

Validation scenarios

To evaluate the effect of defective pixels on the quantification, simulations were made on real images. Here are the results of these simulations showing the impact of hot and cold pixels on the statistics, as well as the corrected result using the median filter method. The images present the visual effect, the absolute values and the error percentage from the original value of these modified images.

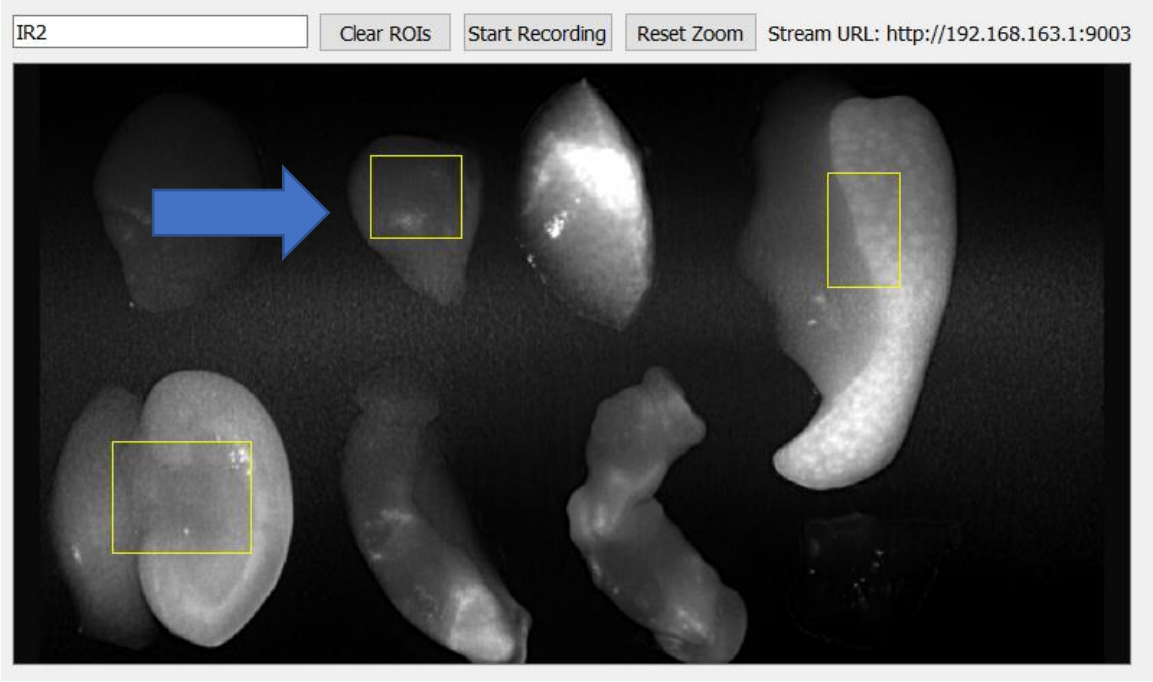

Figure 7 Simple bio distribution image captured in the CondorViewer software. The blue arrow indicates a square ROI that will be used for the following discussions.

The selected ROI has a fairly low signal (mean value of around 531/4095 as shown in Table 4 below). Consequently, it is expected that the hot pixels will have a greater effect than the cold pixels on this particular case. This is acceptable, as the goal of this example is to show the general effect of the defective pixels. Inversely, for higher signal images, the effect of cold pixels would be higher.

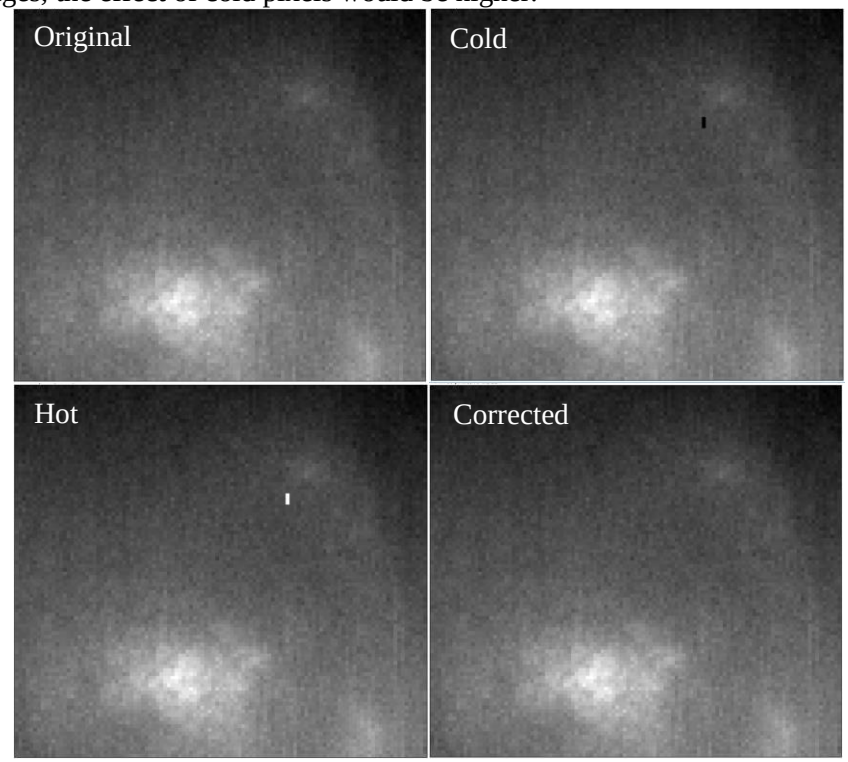

Figure 8 From left to right, top to bottom: Original zoomed in view of the pointed ROI; Cold pixel simulated image; Hot pixel simulated image; Median-filter corrected image. In these images, hot/cold pixels have been simulated by introducing three white/black pixels. This amount of defective pixel for this size of ROI is an extreme case and should not happen in real life. The statistics extracted from these samples should be considered as worst case scenarios.

Data				
	Original	**Cold**	**Hot**	**Corrected**
Mean	531.476	531.345	532.452	531.481
Std Dev	80.905	81.372	99.883	80.903
Mode	513	513	513	513

Difference with original (%)				
	Original	**Cold**	**Hot**	**Corrected**
Mean	0.0000%	0.0246%	0.1836%	0.0009%
Std Dev	0.0000%	0.5772%	23.4571%	0.0025%
Mode	0.0000%	0.0000%	0.0000%	0.0000%

Table 4 Quantification data of the previously presented images, with respective errors.

As seen in these images, hot and cold pixels don't have a huge impact on statistics aside the standard deviation. However, the hot pixels are very distractive to human eyes as they really stand out because of their high contrast. Also, considering that a good identification and correction method is used, there are no real drawbacks of correcting those pixels. Taking all that into account, it has been decided that it was better to identify and correct the defective pixels rather than letting them be.

Software workflows

*** By default, the software will always process the defective pixels. To change this behavior and produce absolute raw images, a workaround is to empty or remove the defective pixels files.**

At application startup, the software will look in the executable folder for defective pixels files. If any files are found, their content will be read and all the defective pixels listed will be fixed on every image that is received from the camera.

When the user wants to modify the pixel lists, the official SOP in the Condor Suite Installation Guide must be followed. Once the pixels are identified by the software, a new csv file containing the newly identified pixels will be saved on the computer, overwriting any previously saved files. Those new pixels will only take effect after the application is stopped and launched again.

Appendix B – Arroyo Laser control

Definitions:
Arroyo: The company that produces our laser system.
Tec source: Arroyo instrument that controls the laser temperature.
Laser source: Arroyo instrument that control the laser diode output.

Compatibility:
The current software is only compatible with Arroyo instruments. The communication is a simple USB serial communication where Arroyo commands are sent to the laser source and tec source. The useful documentation can be found in the Arroyo computer interfacing manual. A copy of this document is also available in the CondorSuite documentation folder.

Configuration:
In a goal of keeping the lasers safe, a lot of thought have been put in the configuration and connection process. When trying to connect a laser, everything in the configuration must be matching and no errors should be returned by the instrument when sending the default values. Any mismatch or error will result in disconnecting the instruments and disabling the laser control in the software.

1. Configuration files
   For a single laser system, both a laser source and a tec source must be connected and available. The configuration files tell the software which tec source is expected to work with which laser. And which laser corresponds to which wavelength. In addition, the remaining of the configuration file contains the default values and command to send to the instrument on connection.
   These files are originally in the csv format for easy editing. Then, via the hardware menu, a user can upload a new csv config file. The file can either be sent as is to the instrument for testing or it can be sent and saved as default. Once the file is saved as default, it is encoded and will be used by the software to configure the lasers. The encoding consists of a basic base64 encoding, which is simple and obfuscate the content enough for our usage.

2. Connection high-level workflow
   When connecting a laser system, here's the main steps that the software takes.

   a. Connect to all available Arroyo instruments
   b. Open config file for each laser until the correct wavelength is found
   c. Get tec serial number for the laser from the config file
   d. Check if the tec source is available
   e. Get the laser SN from the tec config file and compare to laser
   f. If everything matches, send the default values (found in config file) to both instruments

Laser controller (DLL):
This DLL manage the serial connection and disconnection to the instruments and exposes only the useful methods and commands to successfully communicate with the arroyo instruments. Three main classes can be used:
COMInstrument: This is the lower level class allowing to send commands and query to any arroyo instrument. It also exposes non-instrument-specific commands like GetSN and GetModel.
LaserSource: This class is simply a class more specific to the Arroyo laser source.
TecSource: This class exposes methods specific to the Arroyo tec source.
ArroyoLaserController: This class is responsible for connecting, monitoring and controlling the lasers. It reads the configuration files, connect the instruments and sends the default values. Then, it checks the laser temperature every second and make sure it stays in tolerance. If something is wrong, it will disconnect the instruments and emit an alert signal.

Other helper classes:
GenericCommands, LaserCommands and TECCommands: Three static class acting as string enumerations of the instrument commands.

Appendix C – Other Laser control

The other types of laser currently supported (Nawoo Lasers, K172 2 and 4 wavelengths lasers) are much simpler to control as there is no possibilities to change their settings and they don't expose their temperature controller as the Arroyo lasers did. Therefore, their config file is simple. It only consists of the COM port to use and their maximum current value.

Their controllers are also simple, they only require a Serial Communication interface to send the commands as well as a command list of some sort. Also, these types of lasers use the same COM port for all wavelengths, so the SerialComObject is created one level higher and it is passed by reference in the controllers' constructors. Additionally, the wavelengths all share the same config file, so only the first wavelength' configuration is available in the hardware settings. The configuration will be sent to all wavelength when updating it.

The current laser system in use can be changed in the Hardware Settings menu.

Appendix D – Speech Recognition

Description

The speech recognition components are contained within the *VoiceRecognition* project. The main class that the software use externally is *VoiceRecognizer*. This class manages the voice recognition library initialization and recognition. The class itself manages everything and only emits specific signals when a known command is recognized. For an example on how to use it, refer to the class boiler plate comment.
The recognition uses CMUSphinx. More specifically, pocketsphinx, which is a convenient C++ interface to the sphinxbase library. More information can be found here: https://cmusphinx.github.io/

Language, grammar and commands

The library requires multiple resources in order to recognize a language. This library needs

- A dictionary, containing the words and their pronunciation
- An acoustic model, containing the words occurrences and other statistics on the language
- A language model or grammar, defining what the library should look for when listening

For the two first components, the generic dictionary for the English language provided with pocketsphinx are used This allow the recognizer to be able to recognize any words and building them from scratch is quite complicated.

For the third component, which is more application-specific, the generic English language model was tested. The work of deciphering and recognizing the commands would then have to be done by hand by our development team. Unfortunately, the recognition was not good and was thrown off by accents and what not.
Instead of using the whole English language for recognition, a static grammar is used. This grammar follows the JSpeech Language Format (JSGF). It allows us to be very specific about the expected commands and therefore helps the recognizer giving better results.

All these files are in CondorViewer/Resources/Voice.

Updating/Adding commands

When adding commands, these steps should be followed:

1. Update the commands.gram file and add the desired command(s)
2. In VoiceRecognizer.h:
    - Add the new words to the Keywords class
    - Add the proper signals to reflect the command's purpose
3. In VoiceRecognizer.cpp:
    - Modify the processCommands method in order to recognize the newly added command and emit the desired signal
4. In the CondorViewer project, the newly added signal should be routed to the proper component to trigger an action
    - It is a good practice to simply trigger an action on the GUI. This way, it will trigger the same code that would normally be triggered by a user input.

Appendix E – XMP Metadata tools

Description

The software uses the Exiv2 library in order to manage and write XMP metadata to the images file. This metadata includes all sort of information as the camera settings, the lasers settings and other useful information.

Custom tools

In a goal to help the user visualize this metadata, various visualization tools have been implemented.

- First, the metadata tab in the settings dialog is the first way to view the metadata in the software, before even capturing the images. Once the image is captured, it is an inconvenience to rely on the Condor Viewer for the metadata because the users don't have it installed on their computers.
- Second, we developed a python ImageJ plugin. The plugin is in the "tools" sub directory of the repository. Once installed in ImageJ, the user simply must open the image in ImageJ and run the "XMP Reader" plugin. A window will open displaying all the metadata fields for the currently selected image.
- Third, a simple executable has been developed. It must be accompanied by the Exiv2.dll in order to run. To use it, the user simply as to drag and drop the image file onto the executable. A console window will open displaying the metadata fields.