XPIWIT - An XML Pipeline Wrapper for the Insight Toolkit

Documentation

Version 1.0

Andreas Bartschat, Eduard Hübner, Markus Reischl, Ralf Mikut, Johannes Stegmaier

Karlsruhe Institute for Technology (KIT), Institute for Applied Computer Science (IAI)
E-Mail: johannes.stegmaier@kit.edu

June 24, 2015

Contents

List of Figures

Listings

List of Abbreviations

ITK . . . . . Insight Toolkit

XML . . . . . Extensible Markup Language

XPIWIT . . . . . XML Pipeline Wizard for ITK

1 Introduction

1.1 Background and Context

XML Pipeline Wizard for ITK (XPIWIT) is an XML pipeline wrapper for the Insight Toolkit (ITK). It is designed as a pure C++ application based on the ITK and the Qt libraries. In essence, XPIWIT is a small executable with the ability to handle ITK filters and images dynamically without the need to recompile new image analysis pipelines as in usual ITK-based applications. XPIWIT comprises a command-line tool with an Extensible Markup Language (XML) interface to setup image analysis pipelines in a convenient flexible way. Due to the pure C++ implementation, fast processing of multidimensional image data is possible and it was successfully tested to analyze terabytes of image data on distributed computing environments [1]. The main focus of the provided set of filter wrappers is put on processing of biological images. This includes two dimensional images as well as three dimensional image stacks obtained from technologies like confocal, light-sheet or electron microscopy [2, 3]. Of course, XPIWIT can also be used for different image content as well and the implementation of additional filters is straightforward (Sec. 6).

XPIWIT is able to execute filter pipelines of wrapped ITK and custom ITK-like filters. The interface of the pipeline definition is based on XML which gives the user the possibility to create and manipulate filters and filter parameters by hand without the need of compiling the software. The XML interface also provides the option to create the pipelines with scripts and third party tools (e.g. Gait-CAD as described in Sec. 7) or to by a graphical front-end as described in the quick start guide. The start arguments for XPIWIT can be passed either by command line arguments or in a piping mode. All outputs of XPIWIT are images or meta information stored as CSV files. Thus, XPIWIT uses standardized file formats only and can be integrated in a wide range of environments. It is designed to run on Windows, Linux and Mac OS X. Furthermore, it can be used on a single PC as well as on distributed systems like Apache Hadoop and Acquifer's HIVE Technology [4].

1.2 Quick Start Guide

To facilitate getting started with the use of XPIWIT, we prepared precompiled binaries for the latest versions of the three major operating systems. XPIWIT is designed as a command line tool. It has a list of parameters to define input images, the pipeline, the output path and further settings. The filter pipeline itself is described in an XML file. With these inputs, XPIWIT will process each given image with the defined pipeline and create intermediate and final results. In the binary folder of the repository hosted at `http://www.bitbucket.org/jstegmaier/xpiwit`, you find the installation packages of XPIWIT for Windows (tested on versions 7, 8 and 8.1), Ubuntu (tested on versions 12.04 and 14.04) and Mac OS X (tested on version 10.10). The following steps give you

a first impression of how to use XPIWIT and let you process some of the example XML pipelines on exemplary image data:

1. Extract the provided archives in a folder of your choice. There are three sub-folders that are named `Bin`, `Documentation` and `Examples`.

2. The `Bin` folder contains the executables of XPIWIT and the XPIWITGUI that are used to process and generate XML pipelines. Furthermore, the folder contains the shared libraries that are required by the applications. To start XPIWIT or the GUI, use a terminal window on Unix systems or a command prompt on Windows and start the respective application via `./XPIWIT` or `XPIWIT.exe`, respectively.

3. To provide XPIWIT with the information about input, output and XML pipeline to process, use the < operator to pipe an input text file to the executable (Lst. 1). On Unix this would look like `./XPIWIT < myinputfile.txt` and analogously on Windows `XPIWIT.exe < myinputfile.txt`.

4. The example folder provides a set of exemplary XML pipelines and some example images that can be used to test the processing. Furthermore, you will find batch files there, to directly execute the respective examples with the corresponding input paths. For instance, the median filter example (Lst. 8) can be run by calling `./MedianFilter.sh` or `MedianFilter.bat` from the command line. The output images are written to the folder `Examples/Results/`. You can simply open the batch files and the called XML pipelines in the text editor of your choice, to get an impression how the respective commands look like or to manipulate the input and output paths.

5. An overview of all available XML filters can be generated with the XPIWIT executable using the command `./XPIWIT --filterlist myfilters.xml` in the terminal of a Unix-based system or `XPIWIT.exe --filterlist myfilters.xml` in a Windows command prompt. An extensive description of all available XML options and how to consistently link the individual filters to each other is provided in the online documentation hosted at `http://www.bitbucket.org/jstegmaier/xpiwit`. You can simply combine the listed filters to create custom XML pipelines using a text editor and by supplying the desired XML file and IO arguments to the XPIWIT executable using the input files.

6. Alternatively, you can specify a new XML file using the graphical user interface (Fig. 1). Start the GUI using the provided executables, *i.e.*, run `./XPIWITGUI.sh` from a Unix terminal or simply double-click `XPIWITGUI.exe` from a Windows machine. To create a new pipeline simply drag and drop the desired filters from the filter list to the drawing area and connect the respective input and output ports to a reasonable pipeline and adjust the filter parameters by selecting the desired filter. To remove filters or connections from the pipeline, simply perform a right click on the respective item. As a first step, open one of

the example XML files in a text editor and try to build a copy of it using the GUI. Once all filters are appropriately linked and the parameters are adjusted properly, simply drag and drop input and output paths to the respective edit fields in the GUI and press the *run* button. The GUI directly executes the specified pipeline, takes care of interfacing XPIWIT correctly and stores the output data into the specified output folder. The generated XML pipeline can subsequently be saved to process larger jobs, *e.g.* , on a computing cluster.

7. If the operating system of your choice is not compatible with these packages, XPIWIT can also be compiled from the provided sources using CMake and detailed built instructions for this are provided in the online documentation. If you observe any difficulties in getting the code to run, do not hesitate to contact us.

8. **Hints**:

   - Some of the example images are 3D images or have a dynamic range of 16 bit, *i.e.* , make sure to open them with an appropriate image viewer, such as Fiji[1] [5], ICY[2] [6], BioImageXD[3] [7], Vaa3D[4] [8], ParaView[5] [9] or Gait-CAD[6] [10] to name but a few.
   - Results in the result folder are simply overwritten without any notice, *i.e.* , if you do not want your results to be overwritten, either copy your results to a different destination or change the output folder.
   - On Unix systems, the executable mostly does not have execution permissions by default. To change the permissions of the executables use the command `chmod -R 777 *` in a terminal window with the active folder being set to the `Bin/` directory.
   - Folder paths should not contain spaces or special characters. If you observe errors related to missing XML files or if no output is written to the result folder at all, check the path of the location XPIWIT has been installed to.
   - All input paths of XPIWIT should use the "/" character for folder separation. Furthermore, the output path needs to be terminated with a final "/" character. If you manually edit the respective paths, make sure these requirements are met.
   - If you want to import a series of 2D images as a 3D image stack, simply insert the usual ImageReader and Drag&Drop the first file of your image series to the input edit field. Replace the digits of the index of your series file name e.g. by %04d for a four digit index padded with zeros (e.g. `myfile_001.tif` → `myfile_%03d.tif`). The start and end index of the series can be defined in the ImageReader filter settings.

[1] `http://fiji.sc/Fiji`
[2] `http://icy.bioimageanalysis.org/`
[3] `http://www.bioimagexd.net/`
[4] `http://home.penglab.com/proj/vaa3d/home/index.html`
[5] `http://www.paraview.org/`
[6] `http://sourceforge.net/projects/gait-cad/`

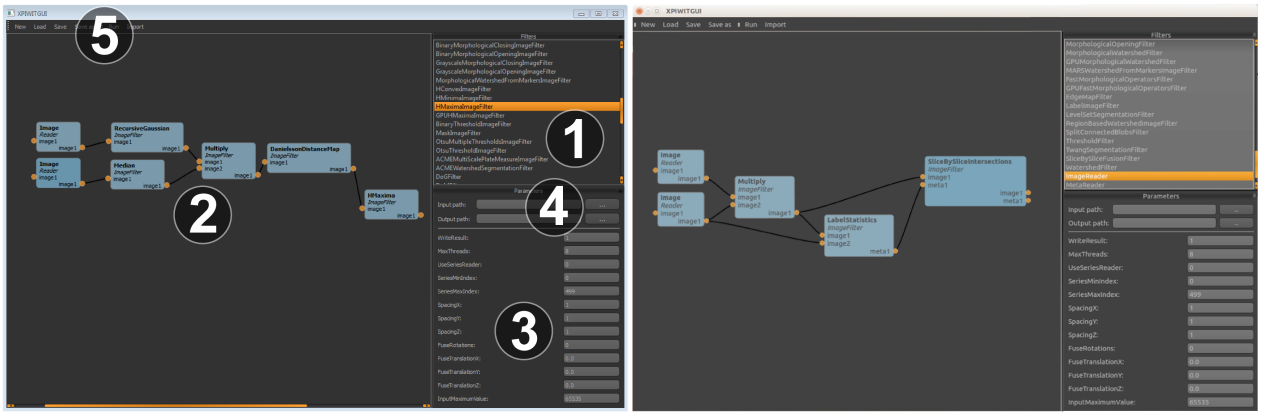

Figure 1: Exemplary screenshot of the XPIWIT GUI on Windows (left) and Ubuntu (right). The main control elements of the GUI are the filter list (1), the pipeline drawing area (2), the parameter adjustment panel (3), the input/output definition panel used for pipeline test runs (4) and the usual controls for saving, loading and executing pipelines (5). Generated pipelines are independent from the GUI, e.g., to perform large-scale analyses on computing clusters.

1.3 Outline

Although the quick start guide might already be sufficient for many users, the remainder of this documentation details all relevant aspects of XPIWIT that are needed to extend its functionality or to use special command line parameters. The commands and settings of XPIWIT are described in Chap. 2. An overview of the XML scheme is given in Chap. 3. Some examples and tutorials on how to use XPIWIT are explained in Chap. 4. Chap. 5 and Chap. 6 describe how to compile XPIWIT and add filter to the sources. An interface to the open source toolbox Gait-CAD is described in Chap. 7.

2 Input Handling

2.1 Introduction

The commands for the execution of XPIWIT can be entered either by command line arguments or by piping. Both ways provide the same functionality. If XPIWIT is called with command line arguments it will only process these command line arguments. If it is called without any parameters, it will process only piped inputs. It is not possible to combine the two input methods. The order of the commands is not relevant and unknown commands will be ignored. Every command consists of the command itself followed by a list of parameters. The parameters are separated by a comma and a whitespace. For the command line input each parameter list has to be placed between quotation marks. In pipe mode XPIWIT will wait for commands until the `end` command is entered. So this has to be the last command when in pipe mode. This command has no meaning when in command line mode. The pipe mode is for example useful in combination with Apache Hadoop Streaming. An exemplary input text file used to start XPIWIT is summarized in the following listing:

Listing 1: Exemplary XPIWIT Input Text File.

```
1 --output ../MyDataSet/Processed/, result
2 --input 0, ../MyDataSet/myimage_t=0012.tif, 3, float
3 --xml ../MyDataSet/mysegmentation.xml
4 --seed 792
5 --lockfile on
6 --subfolder filterid, filtername
7 --outputformat imagename, filtername
8 --end
```

Important Note: File paths have to be in **Unix format**. Also on Windows use the ”/” as the folder separator. Input and output folder paths always have to be terminated by a ”/” character. If XPIWIT fails to produce output data, make sure the path names are spelled correctly.

2.2 Command Overview

2.2.1 Basic Image Processing

input: Defines the input images and metafiles.

output: Defines the output path for the results.

xml: Defines the full path of the xml pipeline document.

logging: Define if logging is saved in a file or printed to command line.

end: Stops waiting for commands and starts the processing, pipe mode only.

2.2.2 Data Handling

subfolder: Creates subfolder for every filter output.

outputformat: Defines the output naming of images.

2.2.3 Metadata

metaDataHeader: Add an additional header line to every csv file.

metaDataSeparator: Defines the separator character for the csv files.

metaDataDelimiter: Defines the delimiter character for numbers.

2.2.4 Extended Features

filterlist: Creates a XML with all supported filter.

lockfile: Creates a lock file while processing.

2.3 Basic Image Processing

2.3.1 Input

The images that should be processed by XPIWIT are defined by *--input*. The input can be a single image file, a txt file with a list of images or a folder with images. If more than one image is selected, all images have to be of the same dimension. Each of the given images will be processed with the given pipeline and parameter set. The *--input* command has three parameters. The first is the number of the image which is mapped to the XML *input* tag. The second is the full file name to the image, the txt-file with the list of images or a path to a folder with images. These files will all be sequentially processed by XPIWIT. The third parameter defines the dimension of the images, it can either be a 2 for two dimensional or 3 for three dimensional images, respectively. Also the meta inputs for the images are defined by the input command. If the input is a metadata file the third parameter is ”csv” instead of a number. Every pipeline input must be defined by one *--input* command. So this command can appear multiple times. One for each pipeline input.

- --input
  - < integer > (mapping to xml input)
  - < filename > or < list of filenames > or < folder name >
  - < integer > (dimension) or ”csv” for metadata

2.3.2 Output

The output path is defined by the *--output* command. It can either be a folder or a list of image filenames.

- --output
  - < folder > or < list of filenames >
  - < string > (prefix)

2.3.3 XML

The XML file with the image analysis pipeline is specified by the *--xml* command. It has only one parameter which is the full file name of the xml file.

- --xml
  - < filename to xml >

2.3.4 Logging

The *--logging* command defines whether log files are generated for each pipeline run or logging information are printed to the command line.

- --logging
  - turn it on (default) or off
    - ”false” or ”0” or ”off”
    - ”file” or ”true” or ”1” or ”on”

2.3.5 End

The *--end* command defines the end of the piped arguments and must be the last command when in piping mode. Otherwise XPIWIT will wait for piped commands.

2.4 Data Handling

2.4.1 Subfolder

The *--subfolder* command defines either subfolder should be used or not. Furthermore the naming conventions can be defined either by the filter id or by the filter name or both (will be separated by "_"). XPIWIT will follow the sequence in the given order.

- --subfolder
  - turn it off or define the naming
    - "false" or "0" or "off" (to store results in a single folder)
    - "filterid", "filtername" (to store results in subfolder for each filter, default)

2.4.2 Output Format

The *--outputformat* command defines naming conventions for output files. It consists of four keys. The keys will be separated by "_" and XPIWIT will follow the sequence in the given order. The used prefix key is the second output command parameter.

- --outputformat
  - "prefix", "imagename", "filterid", "filtername" (output file name convention)

2.5 Metadata

2.5.1 Header

This command defines whether a header line is added to the csv files or not.

- --metaDataHeader
  - turn it on (default) or off
    - ∗ ”false” or ”0” or ”off” (to store files without header line)
    - ∗ ”filterid”, ”filtername” (to store files with header line)

2.5.2 Separator

This command defines the separator used in the csv files.

- --metaDataSeparator
  - < character > (default: ”;”)

2.5.3 Delimiter

This command defines the delimiter for numbers in the csv files.

- --metaDataDelimiter
  - < character > (default: ”.”)

2.6 Extended Features

2.6.1 Filter list

This parameter is used to write a list of all available filters with their parameter into the given XML.

- --filterlist
  - < filename to XML >

2.6.2 Lockfile

The *--lockfile* command defines whether a lock file is set or not. This file prevents two or more instances of XPIWIT to be executed in parallel on one machine to process images with the same output folder.

- --lockfile
  - either turn it on or off (default)
    - ”true” or ”1” or ”on” (to create the file)
    - ”false” or ”0” or ”off” (to not create the file, default)

3 Creating pipelines

The interface for the creation of filter pipelines and their parameterization is based on XML. **It is not recommended to write the XML file manually, instead copy and paste filters from the generated filter list and adapt the parameters or use the provided graphical user interface.** An overview of all available XML filters can be generated with the XPIWIT executable using the command `./XPIWIT --filterlist myfilters.xml` in the terminal of a Unix-based system or `XPIWIT.exe --filterlist myfilters.xml` in a Windows command prompt.

3.1 Structure of the XML interface

Every XML consists of the header line which defines the XML version and the encoding used in the file. The root element for a XPIWIT pipeline is ”xpiwit” followed by the child ”pipeline”. This tag consists of several items which defines the filter used in the pipeline. An overview of the basic structure is presented in Lst. 2.

Listing 2: Structure of the XML pipeline definition

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xpiwit>
3    <pipeline>
4      <item item_id="item_0001">
5        ...
6      </item>
7      <item item_id="item_0002">
8        ...
9      </item>
10   </pipeline>
11 </xpiwit>
```

3.2 Structure of the Filter Items

Every filter in a pipeline is defined within an item tag. The item tag itself has an attribute ”item_id” which defines a unique item id for the element. Every id starts with a letter followed by a sequence that may consist of letters and digits of any length. Every item consists of at least three child elements.

name: Defines the name of the filter.

input: Defines the input images and metadata.

arguments: Defines the parameter for the filter execution.

The name of the filter defines the used filter and must match a name of a filter included in XPIWIT. Otherwise the pipeline execution will be aborted. The input tag consists of children of the type image or metadata. Usually, all inputs have to be defined for a proper filter execution. Both the image and the meta tag contain a reference to an output id of a preceding image filter or a cmd defined image or file that needs to be read from disk. The argument tag contains the parameters of the filter. Every parameter tag consists of the name of the parameter which is selected with the attribute ”key” and the value of the parameter defined by the attribute ”value”. An overview of the definition of an image reader is presented in Lst. 3.

Listing 3: Pipeline Item

```
1 <item item_id="item_0001">
2   <name>ImageReader</name>
3   <input>
4     <image item_id_ref="cmd" number_of_output="1" />
5   </input>
6   <arguments>
7     <parameter key="WriteResult" value="1" />
8   </arguments>
9 </item>
```

3.3 Supported Filters

The list of all supported filter can be created with the *filterlist* command (Sec. 2.6.1). It contains an indicator for the i/o behavior as well as all possible parameter with their default value and a short description. The filter can be simply copy pasted from the list into an individual pipeline. Consider that the ids of the items have to be adapted as well as the input ids.

3.4 Connecting Filters

Each item has a unique id for the item and every output created by the item has a number. The images and meta information are counted independently. Every input contains a reference to the item id that produces the output and its number. If a pipeline contains input references which are not created in the pipeline, the pipeline cannot be processed properly. Lst. 4 contains an example of two connected items. The first item reads an image from disk (specified by the `cmd` indicator as input id reference). The second item takes this image and applies a Gaussian filter to the image. The important line is 17, where the item reference is set to the image reader.

It is possible to give two or more items a reference to one output image. The output image will be hold and copied as long as a filter exists that needs the image.

Listing 4: Connecting Items

```
<?xml version="1.0" encoding="UTF-8"?>
<xpiwit>
  <pipeline>
    <item item_id="item_0001">
      <name>ImageReader</name>
      <input number_images="1" number_meta="0">
        <image item_id_ref="cmd"
          number_of_output="1"/>
      </input>
      <arguments>
        <parameter key="WriteResult" value="1"/>
      </arguments>
    </item>
    <item item_id="item_0002">
      <name>DiscreteGaussianImageFilter</name>
      <input numberImages="1" numberMeta="0">
        <image item_id_ref="item_0001"
          number_of_output="1"/>
      </input>
      <arguments>
        <parameter key="WriteResult" value="1"/>
        <parameter key="Sigma" value="1.0"/>
        <parameter key="MaximumError" value="0.01"/>
        <parameter key="MaximumKernelWidth" value="32"/>
        <parameter key="UseImageSpacing" value="0"/>
      </arguments>
    </item>
  </pipeline>
</xpiwit>
```

3.5 Detailed Tag Descriptions

3.5.1 Name

The name tag holds the name of the filter and is used to create the filter object at runtime. It must match an implemented filter of XPIWIT, otherwise the pipeline cannot be processed. A list of all filters can be created by XPIWIT with the `filterlist` command (Section 2.6.1).

3.5.2 Description

The `description` tag is optional and provides a small description of the filter. It will be ignored by XPIWIT and is meant as a support for the user.

3.5.3 Image Types

The `image_types` tag provides information on the internal templating of the filter. Most filters have only one type describing the precision of the filter. The `type_name` parameter can be changed to following types:

- float (32 Bit), in a range from 0 to 1 with an accuracy of ca. 7 digits (default).
- ushort (16 Bit), positive integral data type with the range 0 - 65535.

Float should be fine in most cases. But if the system runs out of memory a switch to ushort can be useful. The `type_number` parameter is only a hint for the user which image is affected by the type. It is not necessary and will be ignored. It is no problem to combine filters with different data types in one pipeline. XPIWIT will automatically cast and rescale the images to the given data type.

3.5.4 Input

The `input` tag defines the input images and metadata of the filter. It consist of several (at least one) sub tags. For input images the `image` tag and for metadata the `meta` tag is used. Both consist of three parameters. The `item_id_ref` defines the item id of the item that produces the output that should be used. The `number` defines the output number of the image. As most filters have only one output this parameter is usually 1. The third parameter is optional and a reference to the used data type.

3.5.5 Output

This tag is optional and contains information on the produced output images and metadata.

3.5.6 Arguments

The `arguments` tag contains the parameters of the item. The values of the parameters can be adapted but the keys are fixed and should not be touched. Every parameter has a default value and a short description.

3.5.7 Tag Outline

Lst. 5 shows a complete outline of a filter. Tags marked as green are editable by the user. Orange tags are only for information purpose, changes will have no effect. The red tags should be copied from the filter list and not changed manually.

Listing 5: Tag outline

```
<?xml version="1.0" encoding="UTF-8"?>
<xpiwit>
  <pipeline>
    <item item_id="imageReader">
      ...
    </item>
    <item item_id="gaussianFilter">
      <name>DiscreteGaussianImageFilter</name>
      <description>Gaussian smoothing filter.
        Filters the image with a gaussian kernel defined by variance.
      </description>
      <image_types number="1">
        <type type_number="1" type_name="float"/>
      </image_types>
      <input numberImages="1" numberMeta="0">
        <image item_id_ref="imageReader"
          number_of_output="1"
          type_number="1"/>
      </input>
      <output numberImages="1" numberMeta="0">
        <image number="1" type_number="1"/>
      </output>
      <arguments>
        <parameter key="WriteResult" value="1"/>
        <parameter key="Sigma" value="1.0"/>
        <parameter key="MaximumError" value="0.01"/>
        <parameter key="UseImageSpacing" value="0"/>
      </arguments>
    </item>
  </pipeline>
</xpiwit>
```

4 Examples

The following examples show some use cases and the parameterization of XPIWIT. All files can be found in the `Examples` folder that comes with the binary distributions of XPIWIT.

4.1 Setup

Extract the downloaded folder to your hard drive. The root folder of these examples is "XPIWIT" with its subfolder `Bin`, `Documentation` and `Examples`. The XPIWIT executable and the associated GUI can be found in the `Bin` folder, the `Documentation` folder contains the PDF you're currently reading and image files, XML pipelines and the argument files of the example can be found in the `Examples` folder. To run the examples, the Example folder contains some batch scripts. These scripts can also be manipulated with any text editor. On windows these batch scripts have the extension `.bat` and on Unix-based systems the extension `.sh`. To start off and to check if everything is configured properly, simply start by executing one of the prepared batch files by simply double-clicking the `MedianFilter.bat` on Windows systems or by starting the `MedianFilter.sh` from a Unix terminal application. After successful execution, you'll find the results, *i.e.* , a median filtered image in the `Examples/Results/` folder. The following section describes the content of the text input files used to process the data.

4.2 File Contents

As shown in Listings 6-8, the batch file is used to call the XPIWIT executable and passes the input file to it, such that the executable is informed about input, output and XML file paths. The input file (`MedianFilter.txt` in this case) contains this information, where line 1 is the output folder, line 2 is the input image and line 3 is the XML pipeline to be applied on the input image. If further inputs are required, simply add a second input line with an increased index. Finally, the `MedianFilter.xml` defines the actual pipeline, which is simply comprised by an image reader, an intensity adjustment filter and a median filter. To properly connect the filters within a pipeline, make sure that the respective input `item_id_ref` parameters point to the correct output of a preceding filter.

Listing 6: Content of "MedianFilter.bat"

```
1 "../Bin/XPIWIT.exe" < "../Examples/XMLPipelines/MedianFilter.txt"
```

Listing 7: Content of "MedianFilter.txt"

```
1 --output ../Examples/Results/
```

```
--input 0, ../Examples/Data/3D/TestImage3D.tif, 3, float
--xml ../Examples/XMLPipelines/MedianFilter.xml
--seed 0
--lockfile off
--subfolder filterid, filtername
--outputformat imagename, filtername
--end
```

Listing 8: Content of "MedianFilter.xml"

```
<?xml version="1.0" encoding="UTF-8"?>
<xpiwit>
   <pipeline>
      <item item_id="item_0001">
         <name>ImageReader</name>
         <input number_images="1" number_meta="0">
            <image item_id_ref="cmd" number_of_output="0"/>
         </input>
         <output number_images="1" number_meta="0">
            <image number="1"/>
         </output>
         <arguments>
            ...
         </arguments>
      </item>
      <item item_id="item_0002">
         <name>RescaleIntensityImageFilter</name>
         <input number_images="1" number_meta="0">
            <image item_id_ref="item_0001" number_of_output="1" type_number="
               1"/>
         </input>
         <output number_images="1" number_meta="0">
            <image number="1"/>
         </output>
         <arguments>
            ...
         </arguments>
      </item>
      <item item_id="item_0003">
         <name>MedianImageFilter</name>
         <input number_images="1" number_meta="0">
            <image item_id_ref="item_0002" number_of_output="1" type_number="
               1"/>
         </input>
         <output number_images="1" number_meta="0">
            <image number="1"/>
         </output>
         <arguments>
            ...
```

```
38          </arguments>
39        </item>
40      </pipeline>
41  </xpiwit>
```

Instead of defining the XML manually, it is strongly recommended to use the XPIWITGUI executable, which provides a graphical interface to the XML generation. To get used to the functionality, try to re-assemble the above-mentioned median filter example in the GUI. To do this, perform the following steps:

- Open the GUI and make sure the XPIWIT executable is located in the same path as the GUI and has executable permissions.
- Drag and Drop the required filters from the filter list to the drawing area (ImageReader, RescaleIntensityFilter and MedianFilter).
- Connect the inputs and outputs to form a reasonable pipeline as shown in Fig. 2.
- Pipeline parameters can be adjusted by selecting the respective filter and by editing the parameter in the updated dialog (bottom right).
- To test the pipeline, simply drag and drop an image file or a folder to the `Input1` edit field and analogously add an output folder to the `Output path` edit field.
- By pressing the `run` button, the pipeline will be executed on the specified input image and the results are stored into the selected location.
- You can save the pipeline using the `Save/Save as` buttons. Pipelines stored from within the GUI can be re-opened and edited later on. Furthermore, the corresponding XML and XPIWIT input files are saved and can be applied on different images, *e.g.* , on a computing cluster.

If you succeeded with the example above, try creating a more complex processing pipeline. Examples for such pipelines are provided in the `XMLPipelines` folder and comprise a maximum intensity projection of a 3D input image along all 3 dimensions (`MaximumProjections.*`), the TWANG segmentation algorithm as described in [1] (`TWANGSegmentation.*`) and finally an example that multiplies a raw input image with a binary mask using two image readers (`TwoReaderExample.*`). Screenshots of how the pipelines should look like are also located in the `XMLPipelines` folder.

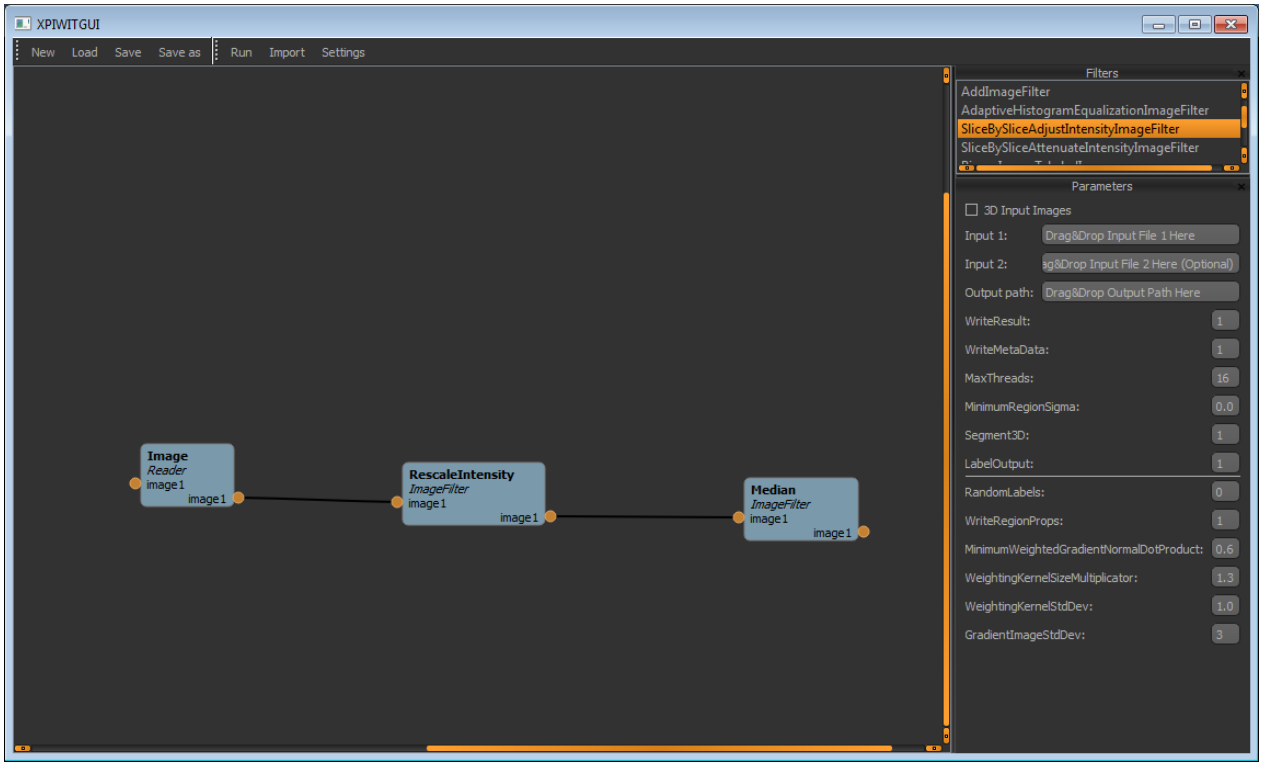

Figure 2: Exemplary XPIWIT pipeline for a median filter.

4.3 Possible Input Path Formats

XPIWIT can handle various formats of the input file names. Besides the use of a single input image, XPIWIT can process entire folders, lists of input images and optionally specify the output names of the generated result images. The following sections provide examples, how the syntax should look like if these formats are used.

4.3.1 Single Input

Processes a single image file with the pipeline and stores the results in the ”Results” folder.

input: Examples/Data/3D/TestImage3D.tif (mapped to ”cmd” with number ”0”)

output: Examples/Results/

xml: Examples/XMLPipelines/MedianFilter.xml

Listing 9: Single input cmd commands

```
XPIWIT.exe --xml "../Examples/XMLPipelines/MedianFilter.xml"
--input "0, ../Examples/Data/3D/TestImage3D.tif, 3, float"
--output "../Examples/Results/"
```

Same parameter for piping mode:

Listing 10: Single input piped commands

```
XPIWIT.exe <
  "../Examples/XMLPipelines/MedianFilter.txt"
```

Listing 11: Content of arguments_image

```
--xml ../Examples/XMLPipelines/MedianFilter.xml
--input 0, Examples/Data/3D/TestImage3D.tif, 3, float
--output Examples/Results/
--end
```

4.3.2 Processing a Folder of Images

Processes all images in the folder ”2D” with the specified pipeline.

input: Examples/Data/2D/ (mapped to ”cmd” with number ”0”)

output: Examples/Results/

xml: Examples/XMLPipelines/MedianFilter.xml

Listing 12: Folder processing with cmd commands

```
XPIWIT.exe --xml "../Examples/XMLPipelines/MedianFilter.xml"' --input "0, ../
   Examples/Data/2D/, 2, float" --output "../Examples/Results/" --end
```

4.3.3 Processing Lists of Images

Processes lists of images defined in a text file. Each line of the input file should contain exactly one reference to an image

input: Examples/Data/input.txt (mapped to ”cmd” with number ”0”)

output: Examples/Results/

xml: Examples/XMLPipelines/MedianFilter.xml

Listing 13: Processing lists with cmd commands

```
XPIWIT.exe --xml "../Examples/XMLPipelines/MedianFilter.xml"' --input "0, ../
    Examples/Data/input.txt, 2, float" --output "../Examples/Results/" --end
```

Listing 14: Content of the input.txt file

```
../Examples/Data/2D/TestImage2D_0000.tif
../Examples/Data/2D/TestImage2D_0010.tif
../Examples/Data/2D/TestImage2D_0020.tif
../Examples/Data/2D/TestImage2D_0030.tif
```

4.3.4 Processing Multiple Input Image

Processes lists of images defined in a text file. Each line of the input file should contain exactly one reference to an image

input 1: Examples/Data/3D/TestImage3D.tif (mapped to "cmd" with number "0")

input 2: Examples/Data/3D/TestImage3D_BinaryMask.tif (mapped to "cmd" with number "1")

output: Examples/Results/

xml: Examples/XMLPipelines/TwoReaderExample.xml

Listing 15: Processing lists with cmd commands

```
XPIWIT.exe --xml "../Examples/XMLPipelines/MedianFilter.xml"' --input "0, ../
    Examples/Data/3D/TestImage3D.tif, 3, float" --input "1, ../Examples/Data/3
    D/TestImage3D_BinaryMask.tif, 3, float" --output "../Examples/Results/" --
    end
```

Analogously, the two inputs could also be two input folders. However, this only works if both folders contain exactly the same amount of images and if the image names are named similarly, such that an alphabetic sorting yields the same order.

5 Compiling the Sources

This chapter will focus on the compilation of XPIWIT and the required third party libraries. On Windows machines this can generally be performed with installed Visual Studio 2012 (express version does not work). For other operating systems such as Linux, XPIWIT comes also with `CMakeLists.txt` files so feel free to use any operating system and compiler you like.

5.1 Compiling XPIWIT on Windows

5.1.1 General Information

- In case you want to be able to process `tiff`-files larger than 4GB it is necessary to have a BigTiff compatible `libtiff` version installed as well as enabling the ITK CMake flag `ITK_USE_64BIT_IDS` during the ITK makefile generation (`http://bigtiff.org/`).
- It is crucial to compile ITK libraries and executables in 64 bit if large images should be used.
- We recommend to store ITK in at most two subfolders to the system root, otherwise path limits from the file system may be exceeded.

5.1.2 Installing the Qt SDK

The Qt libraries can be downloaded from the Qt websites:
`http://qt-project.org/downloads`.

Choose the "Qt Online Installer for Windows". Open the executable, click next and enter your installation path (e.g. C:\Qt or D:\Qt). The settings in Fig. 3 are enough to compile XPIWIT (Qt 5.x.x for Windows 64-bit (VS 2012)) but feel free to install everything you like. Finish the installation by accepting the license and download Qt.

5.1.3 Install Visual Studio 2012

If you want to use Visual Studio 2012 it has to be installed (note that the express version does not work). Qt provides an add-in for Visual Studio that has to be installed as well. You can download it from `http://qt-project.org/downloads` (Visual Studio Add-in 1.x.x for Qt5). After the installation start Visual Studio and browse to Qt5\Qt options in the tab menu. Setup the installed Qt version, *e.g.* , with the name "Qt520_msvc2012_x64_static" as shown in Fig. 4.

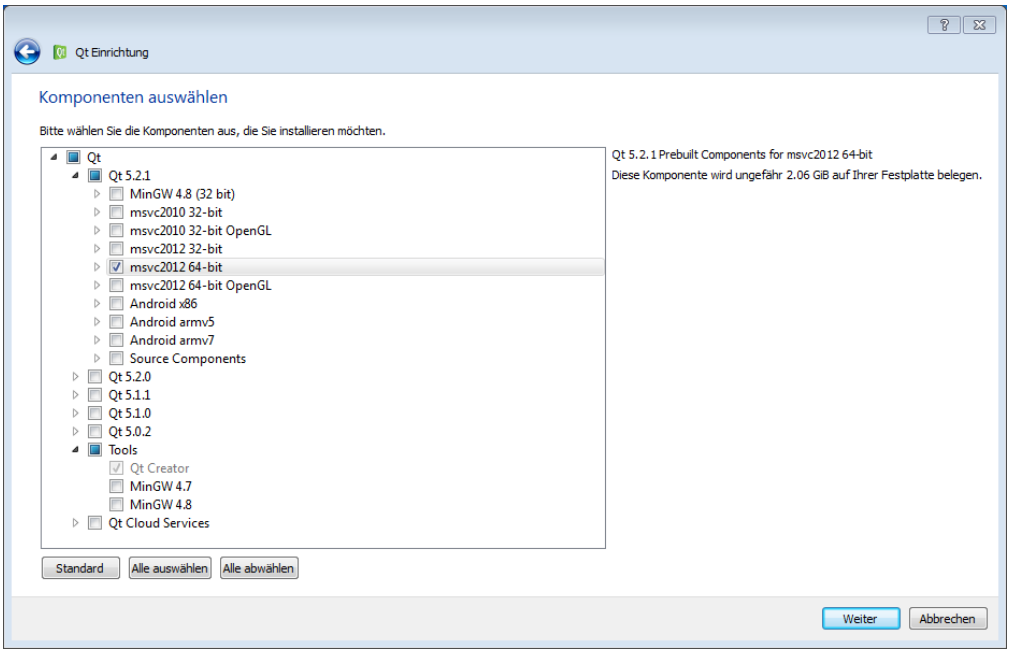

Figure 3: Qt installation

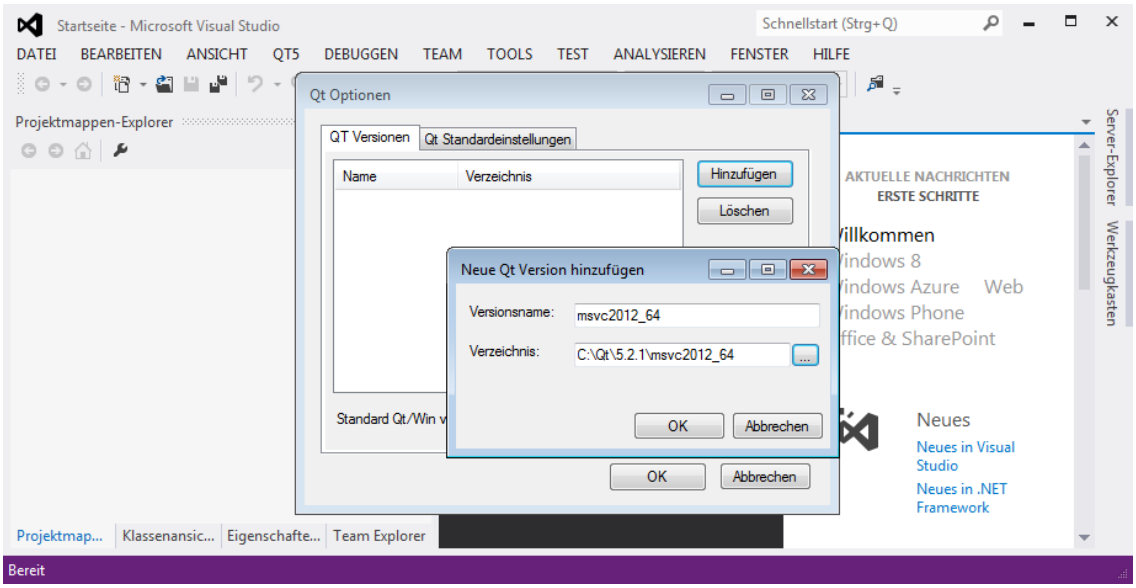

Figure 4: Qt installation

If you do not have a Visual Studio version it is also possible to use any other IDE (QtCreator, Code Blocks) with the Visual C++ compiler or MinGW. Make sure you have the runtime installed if you want to use visual c++ `http://www.microsoft.com/en-us/download/details.aspx?id=30679`.

5.1.4 Compiling the Insight Toolkit (ITK)

- Download and install CMake.
  - `http://www.cmake.org/`
  - Resources (top, right)
  - Downloads
  - Binary distributions (32Bit version will be fine even if you plan to compile XPIWIT in 64Bit)
  - e.g. cmake-2.8.12.2-win32-x86.exe
  - install CMake. (default settings of wizard will work)
- Install ITK sources. (The CMake lists will work with the latest version of ITK but the VS project works with version 4.3.2 only)
  - The version 4.3.2 comes with this installation kit "InsightToolkit_4.3.2.zip".
  - If you want the latest version:
  - `http://www.itk.org/`
  - Resources (top, right)
  - Downloads
  - Library Source (Windows: zip, Linux: tar.gz)

Now create a folder near the root of any hard disk. We recommend to use something like C:\ITK (or D:\ITK). If you go to your download folder you can right click on the ITK (*e.g.* InsightToolkit-4.3.2) zip folder and select extract all (Fig. 5). Now type in the folder the created folder (C:\ITK) (Fig. 6).

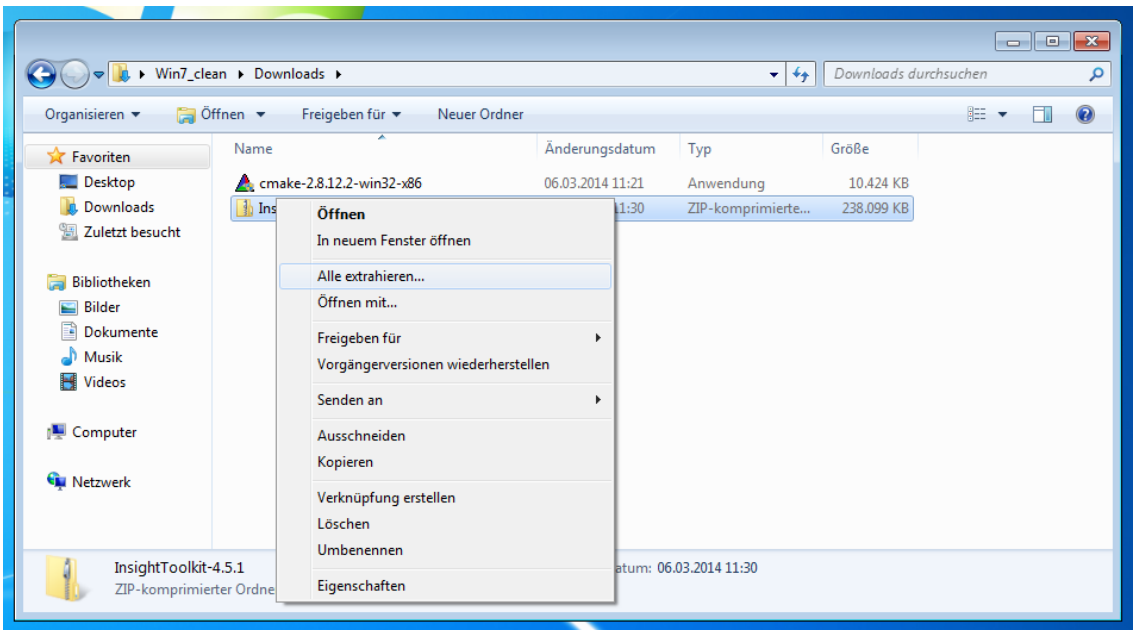

Figure 5: Extract ITK

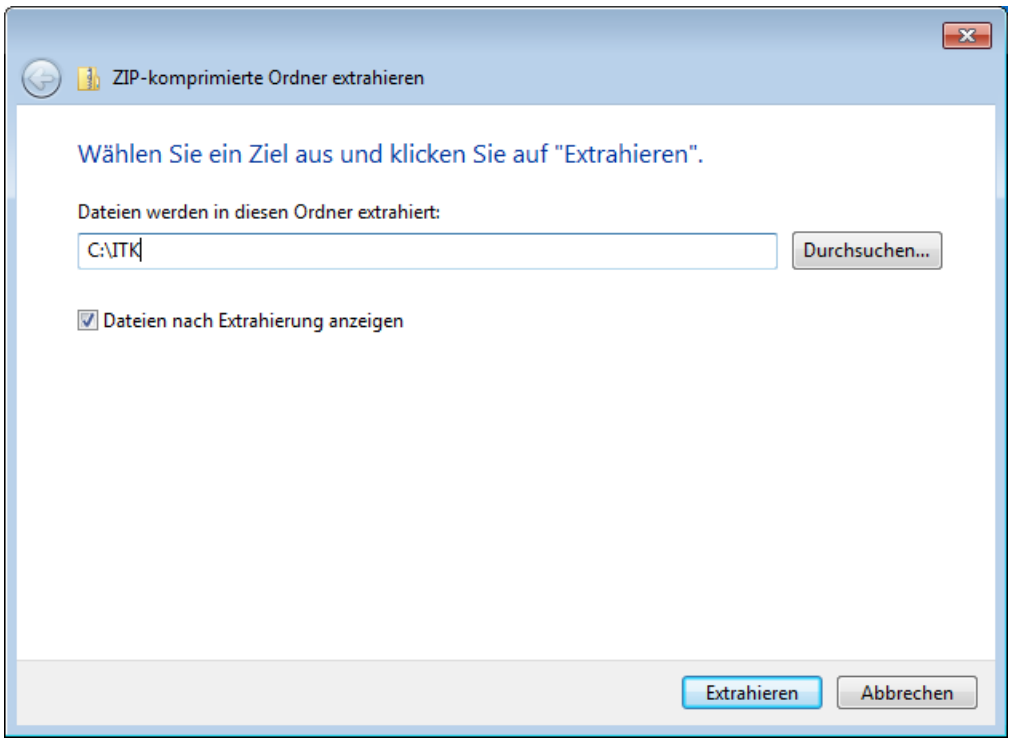

Figure 6: ITK installation path

Your directory structure should now look like (C:\ITK\InsightToolkit-4.3.2)

- C:\ITK\InsightToolkit-4.3.2 with content:
  - CMake
  - Documentation
  - ⋮

Now start CMake-GUI (windows\all programms\cmake\cmake(cmake-gui)). The GUI will show up and you can select the ITK source code with the button Browse Source. Then select a folder to store the binaries. We recommend to use C:\ITK\ITK432msvc2012 (or D:). The configuration should look like the one in Fig. 7.

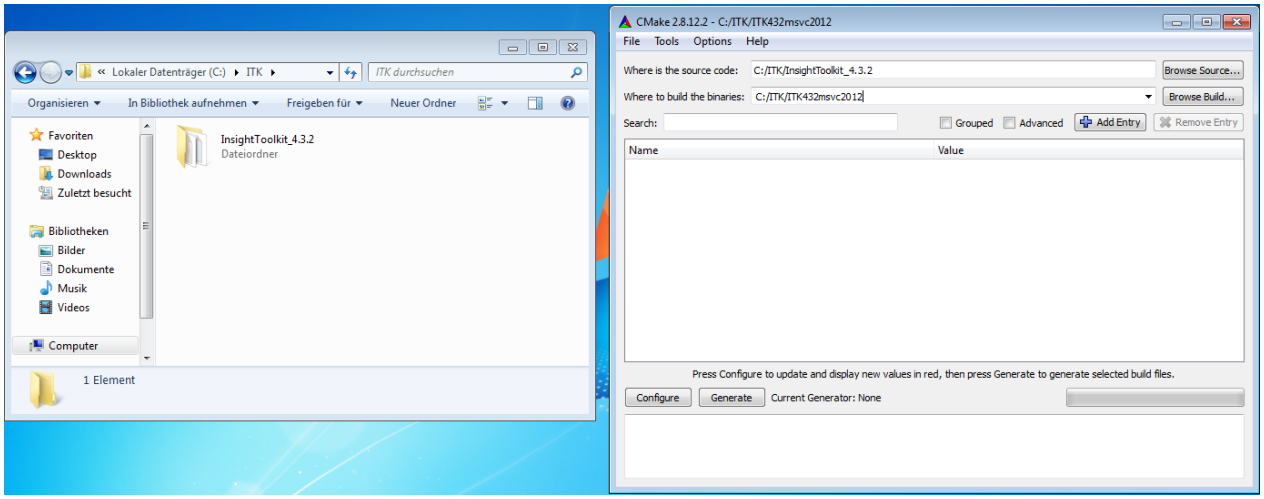

Figure 7: ITK installation

With a click on configure CMake will ask to create the binaries folder if it does not exist and then show a list with all supported compilers. For Visual Studio choose, *e.g.* , ”Visual Studio 11 Win64” as it is shown in Fig. 8. Click finish and the configuration will run.

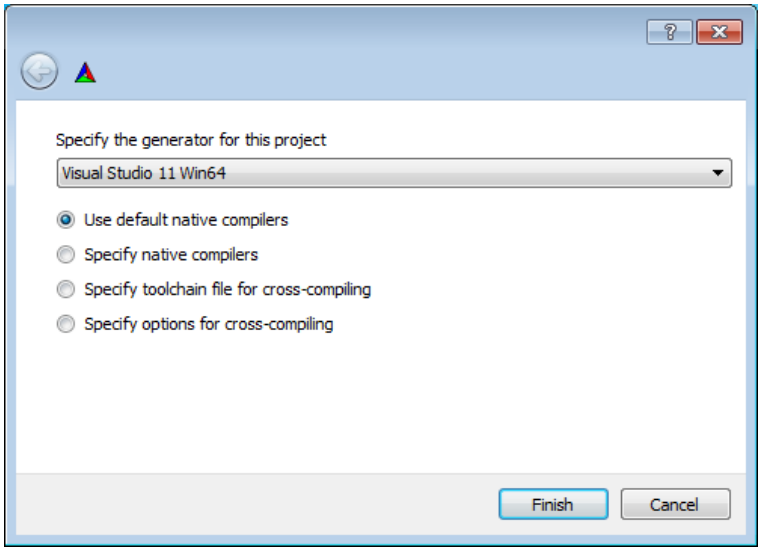

Figure 8: ITK installation

Now it's time to set some flags. Check the box "Advanced" in the GUI and adapt the settings as you like. We recommend changing the following flags (use the search bar):

- Switch to on
  - ITK_USE_64BITS_IDS On
  - ITK_BUILT_ALL_MODULES On
  - ITK_BUILT_DEFAULT_MODULES On (version $>$ 4.3)
  - Module_ITKReview On (version $>$ 4.3)
  - CMAKE_USE_RELATIVE_PATHS On
- Switch to off
  - BUILD_EXAMPLES Off
  - BUILD_SHARED_LIBS Off
  - BUILD_TESTING Off
  - BUILD_DOCUMENTATION Off

Configure the project a second time to set the flags. The GUI should now look like the one in Fig. 9.

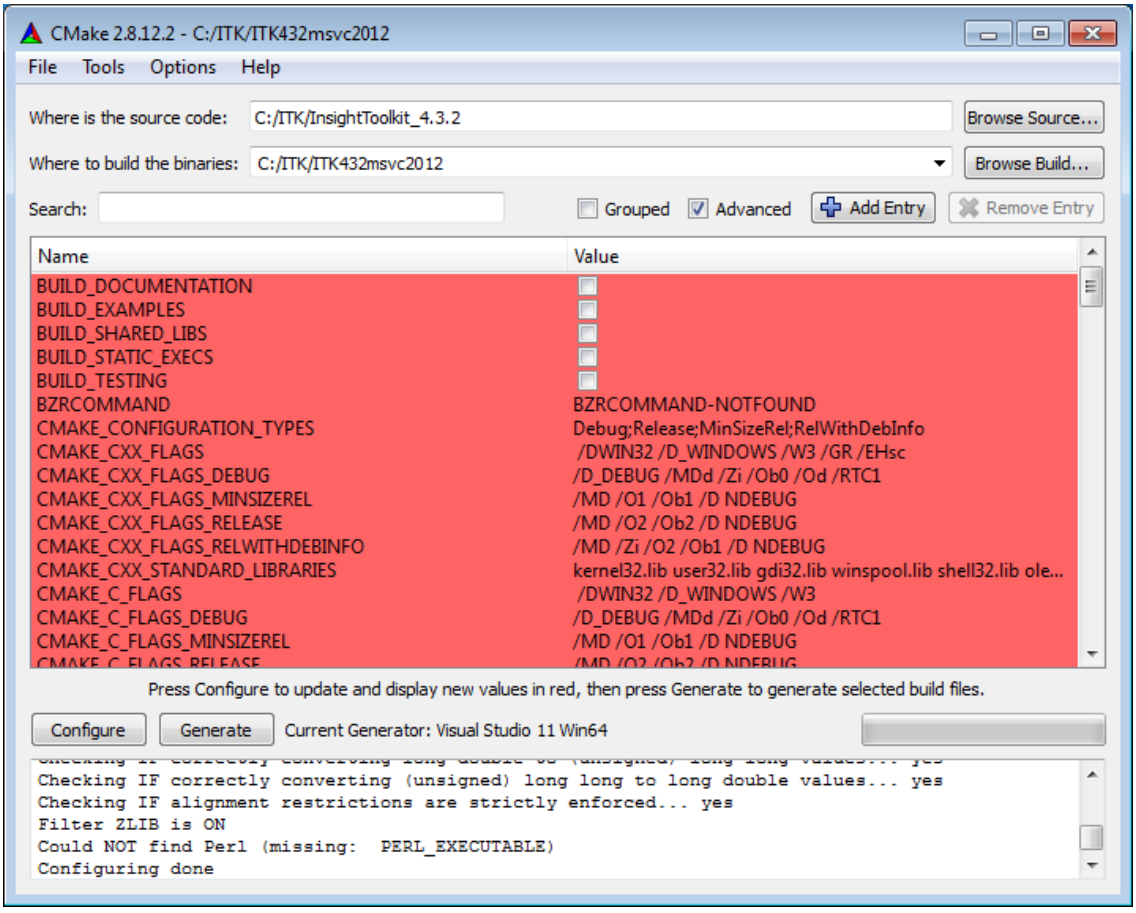

Figure 9: ITK installation

With a click on ”Generate” the project will be generated. After this step is finished the CMake GUI can be closed. Browse to the path ITK binary path (C:\ITK\ITK432msvc2012) and open the ”ITK.sln” project file (Fig. 10). Visual Studio will open and load the project. First of all make sure to use all cores for your compilation. In ”TOOLS\OPTIONS” browse to Projects and ”Solutions\Build and Run” and make sure that the maximum number of parallel project builds is equivalent to the number of your cores. Build the binaries with ”BUILD\BUILD SOLUTION”. This will take a while (from 10 min up to hours depending on your machine). Repeat this step after selecting a release build. See Fig. 11. If no error occurred ITK is ready to use.

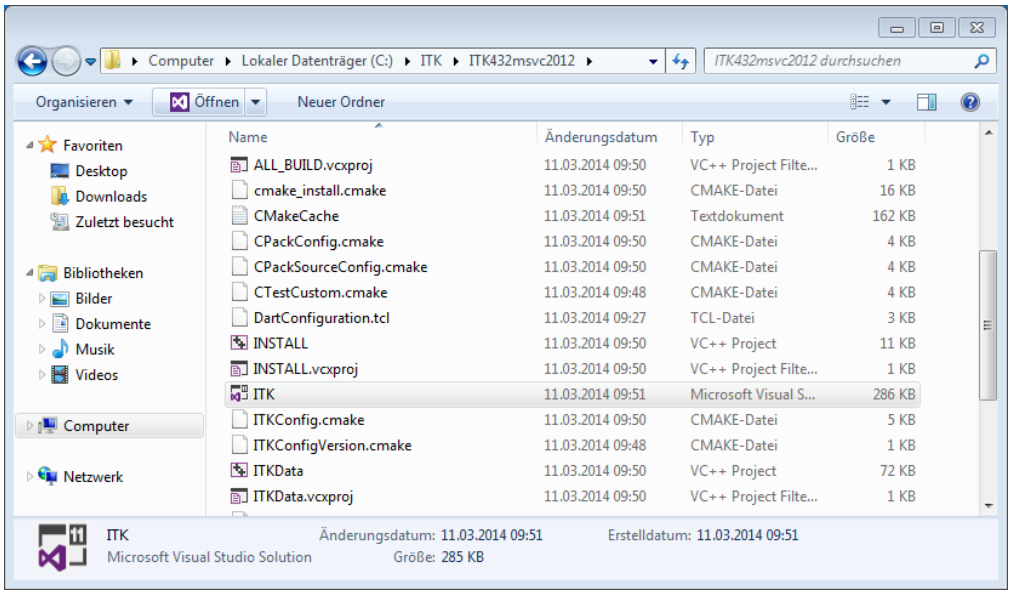

Figure 10: Open the ITK project file.

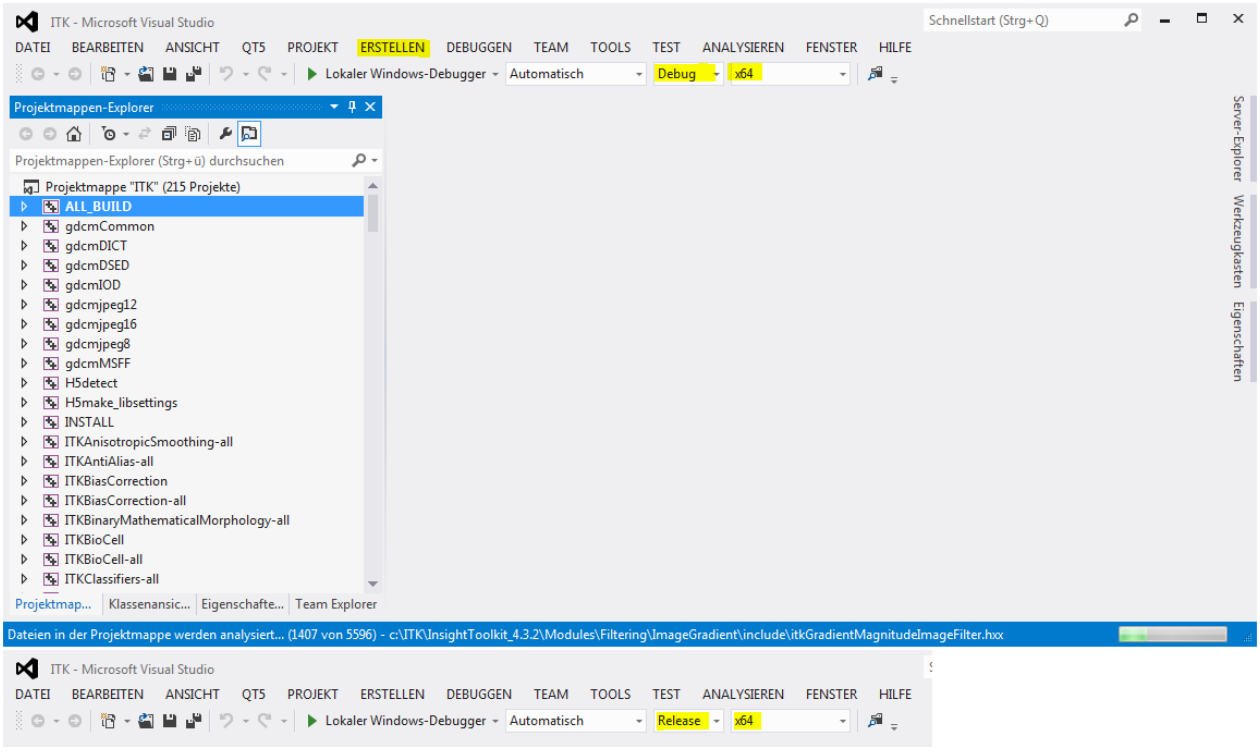

Figure 11: Compile ITK

5.1.5 Download XPIWIT

The source code of XPIWIT is hosted on "Bitbucket" (`https://bitbucket.org/jstegmaier/xpiwit`). It is managed as a GIT repository therefore you need to install a GIT client. It can be downloaded from `http://git-scm.com/`. Keep the default settings of the installer. After the installation open the "Git Bash" and navigate to the root folder where you want to download XPIWIT (linux commands!). For this example we choose "C:" as shown in Fig. 12. With the command git clone a repository can be cloned. Enter git clone "url of repository" and the name of the folder "XPIWIT" in this case (Fig. 12). Now switch to the created folder "cd XPIWIT" and enter "git checkout XPIWIT_0.2" to switch to the branch XPIWIT version 0.2. Fig. 12 shows the commands in the bash.

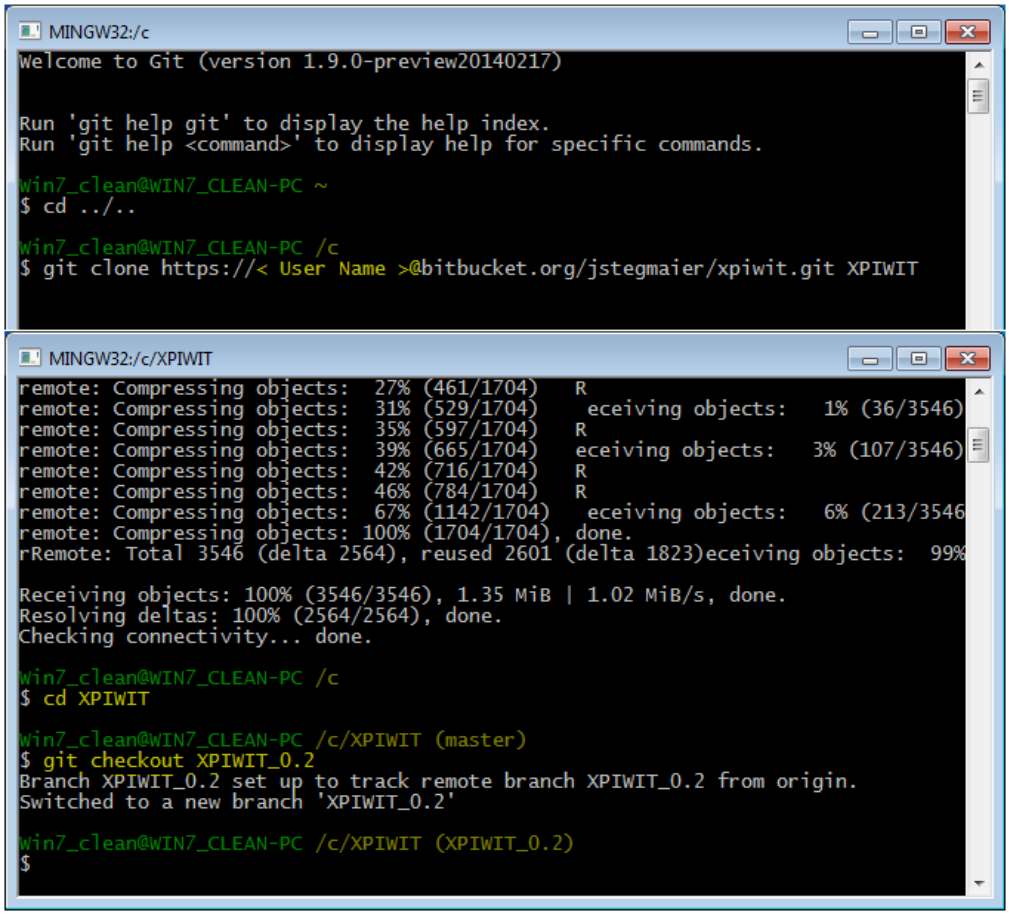

Figure 12: Clone the XPIWIT repository.

5.1.6 Compiling XPIWIT

XPIWIT comes with Visual Studio 2012 project files and with CMake files. So you can choose which one to use.

Set system variables: Open the system overview with the shortcut ”windows + pause” and click on ”Advanced System Settings”. Within the tab Advanced click the button ”Environment Variables”.

To add a variable click new on the top user variables and create the following sets (see Fig. 13).

- Name: ITKDIR - Value: Path to the ITK sources.
- Name: ITKBUILD_VS2012 - Value: Path to the ITK build.
- Name: QTDIR - Value: Path to the downloaded Qt Dir.

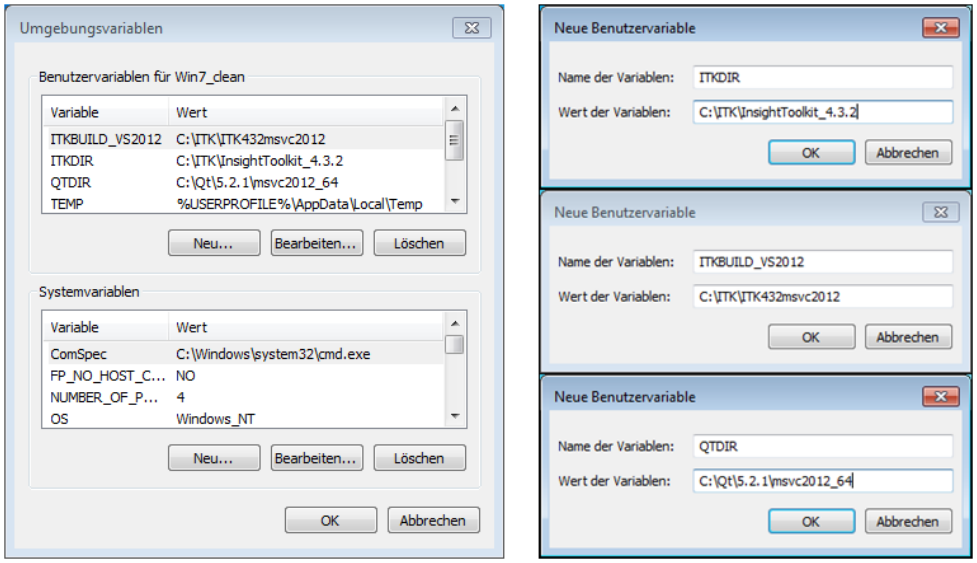

Figure 13: Set system variables

Visual Studio Project: If the system variables are set the Visual Studio project can be found in the XPIWIT repository in ”(root)\XPIWIT\Project\VisualStudio2012\XPIWIT.sln”. Use the x64 configuration to create XPIWIT for debug and release builds.

CMake Lists: To use the CMake lists open the CMake-GUI and set the source path to `(root)\XPIWIT\Project\CMakeQt5` to create the make files or projects for the compiler of your choice. QtCreator, for instance, can also handle CMake lists directly without CMake.

5.2 Compiling XPIWIT on Unix-based Platforms

5.2.1 General Information

We compiled and tested XPIWIT on Ubuntu (12.04, 14.04) and Mac OS X 10.10. Of course, it is up to you which compiler to use. For instance, for Ubuntu we used the following third party applications and libraries:

- gcc 4.6.3 and make 3.81
- CMake 2.8.9
- ITK 4.5.1
- Qt 5.2.1

The remainder of this section presents detailed information about the installation and compilation of the necessary tools and libraries.

5.2.2 Installing the Qt SDK

Get the latest Qt SDK from `http://download.qt-project.org/official_releases/qt/5.2/5.2.1/qt-opensource-linux-x64-5.2.1.run`. To install the file type `./qt--opensource-linux-x64-5.2.1.run` in a terminal window. If permissions for execution are not set properly try `chmod 700 qt-opensource-linux-x64-5.2.1.run` to enable the execution flag.

5.2.3 Install GCC and CMake Build Tools

On Linux systems it is recommended to use the GNU Compiler Collection (gcc, g++ and make). To install the essential parts of the GNU Compiler Collection use the following command in a terminal window `sudo apt-get install build-essential`. More information on the setup procedure can be obtained from `https://help.ubuntu.com/community/InstallingCompilers`.

Download and install the CMake build tool (type: `sudo apt-get install cmake` in a terminal window) and optionally the associated graphical user interface (type: `sudo apt-get install cmake-gui` in a terminal window). If you're using Qt5, make sure the version of CMake is greater or equal to v2.8.9 (the version number of the installed CMake can be obtained via `cmake --version`). If the repository version is lower than v2.8.9, you can get the latest version of CMake from `http://www.cmake.org/cmake/resources/software.html` and build the latest version manually. The CMake GUI can be started using the command `cmake-gui` within a terminal window.

Hint: If your machine has multiple CPUs available you can speed up the compilation using `make -jN` to use `N` cores in parallel for all later build steps.

5.2.4 Compiling the Insight Toolkit (ITK)

Get the latest ITK version from `http://www.itk.org/ITK/resources/software.html`. Extract the ITK sources e.g. into `XPIWIT/ThirdParty/ITK`. To compile the obtained ITK version on your platform, run `cmake-gui` from a terminal window. In the graphical user interface set the source folder to the contents of the extracted ITK sources. Specify an output directory for the ITK binaries, e.g. use the `XPIWIT/ThirdParty/ITK/Buildx64_Linux/` sub-directory in the XPIWIT main folder.

After having specified the input and output paths press `Configure` and choose `Unix Makefile` as a compiler option (See Fig. 14). In the CMake options, check the `Advanced` checkbox and make sure the flags `Module_ITKReview` and `ITK_USE_64BITS_IDS` are enabled. For smaller installations and faster compilation of ITK disable the flags `BUILD_EXAMPLES` and `BUILD_TESTING` (See Fig. 15). Note that a BigTiff compatible version of `libtiff` has to be present on your system, if you intend to use image sizes greater than 4GB.

Once all settings have been adjusted properly, press `Configure` again to update the settings. To create the Unix Makefiles with the current settings, press `Generate`. In a terminal window, change the current directory to the output directory previously specified (e.g. `cd XPIWIT/ThirdParty/ITK/Buildx64_Linux/`) and run the command `make` to build the ITK binaries. This should build the latest ITK version with your desired settings.

5.2.5 Downloading XPIWIT

Get the latest version of XPIWIT by cloning the git repository to your local disk as described in Sec. 5.1.5. If git is not installed on your system yet, use `sudo apt-get install git` to install it.

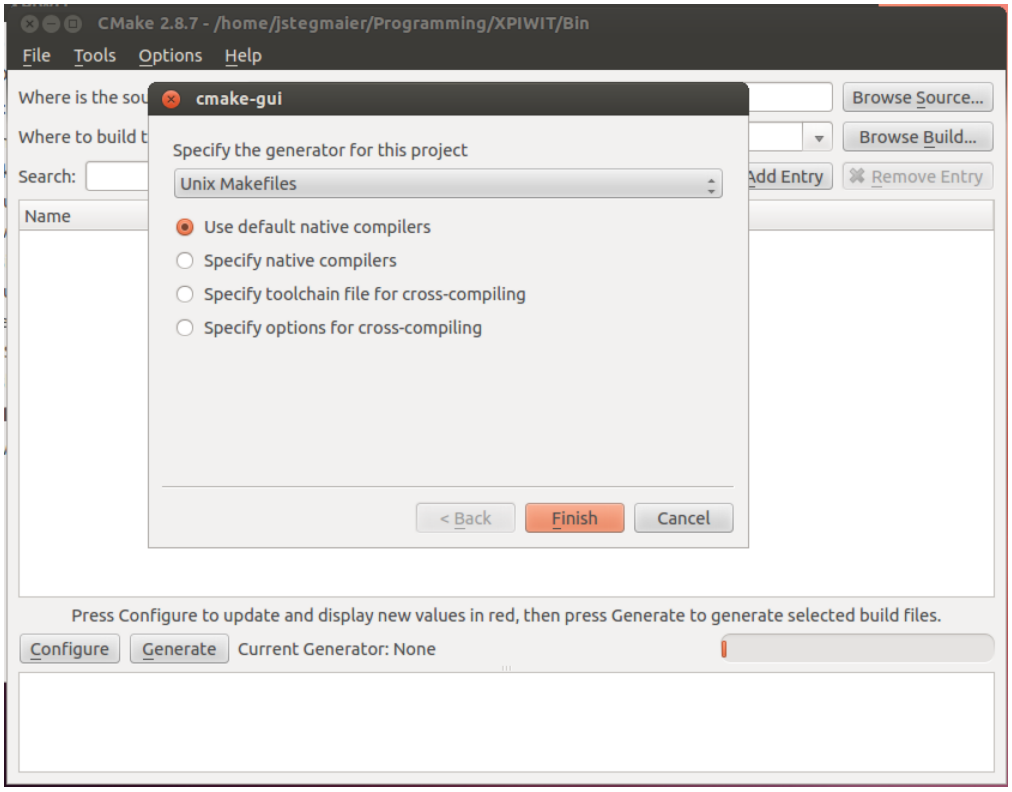

Figure 14: Select Unix Makefile as a code generator.

5.2.6 Compiling XPIWIT

Open a new instance of `cmake-gui` and select the folder `XPIWIT/Projects/CMakeQt5/` for both the source folder and the binary output folder. Press `Configure` and select `Unix Makefile` as a code generator (Fig. 16).

If the directories `ITK_DIR` or `Qt5Core_DIR` are not found automatically, select the appropriate folders (See Fig. 18 and Fig. 17).

To update changes to the CMake settings, press `Configure` again. Once all is properly setup and no errors are left, press `Generate` to create the Unix Makefile. In a terminal window, change the path to `XPIWIT/Projects/CMakeQt5/` and type `make` to build the XPIWIT executable. The compiled XPIWIT binary can be found in `XPIWIT/Bin/XPIWIT`.

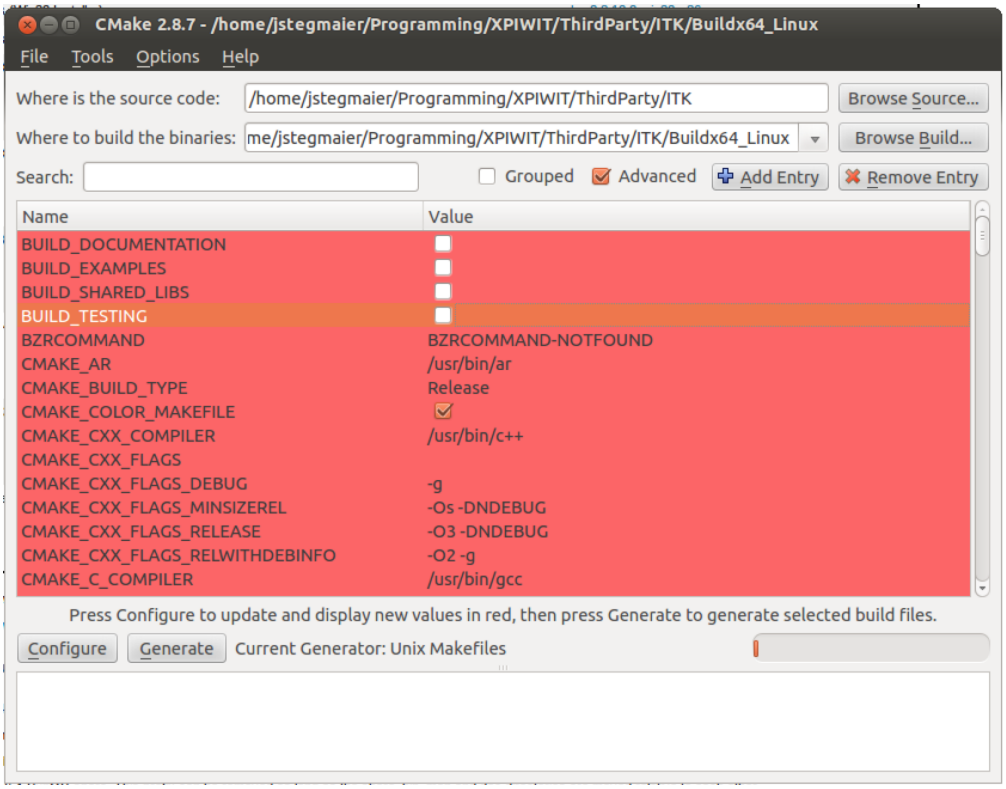

Figure 15: Disable building Examples and Testing if you want to keep compilation times and disk usage small.

5.2.7 Compiling the XPIWIT GUI

The graphical user interface for XPIWIT is compiled exactly the same way as XPIWIT. The `CMakeLists.txt` can be found in `XPIWIT/Projects/CMakeQt5GUI/`. The only additional dependencies are QtGui, QtWidgets and the QNodesEditor (by Stanislaw Adaszewski[7]), and all of them are already added to the `CMakeLists.txt` file. The respective folders should generally be found automatically after specifying the QtCore path. If not, simply redirect CMake to the appropriate locations. The compiled XPIWITGUI binary can be found in `XPIWIT/Bin/XPIWITGUI`. If no filters appear in the filter list of the GUI, make sure a file named `myfilters.xml` that contains all available XPIWIT filters and can be produced as described in the quick start guide.

[7] `http://algoholic.eu/qnodeseditor-qt-nodesports-based-data-processing-flow-editor/`

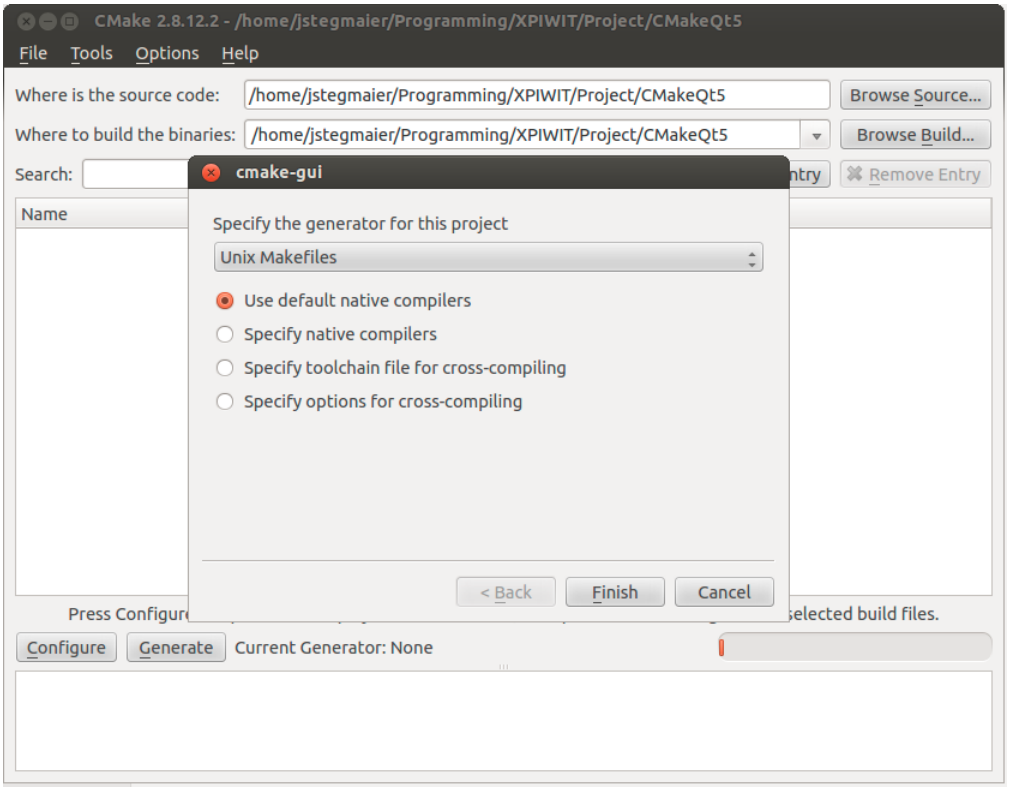

Figure 16: Folder settings and code generator selection for XPIWIT.

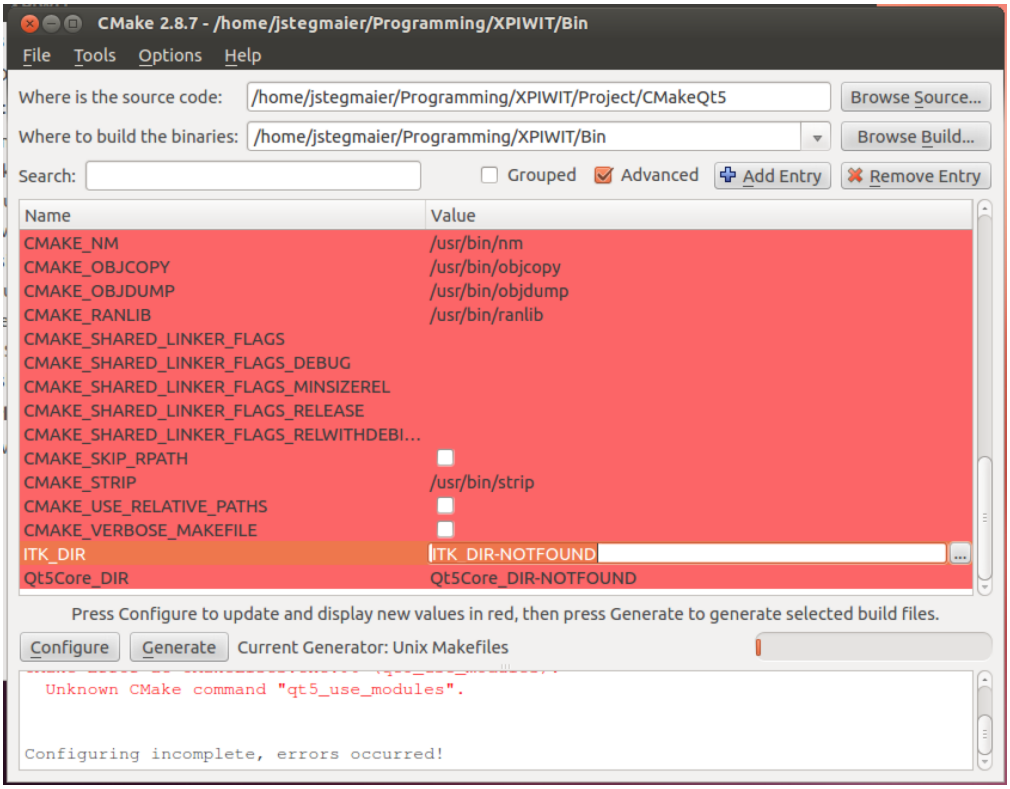

Figure 17: CMake settings if ITK path is not found automatically.

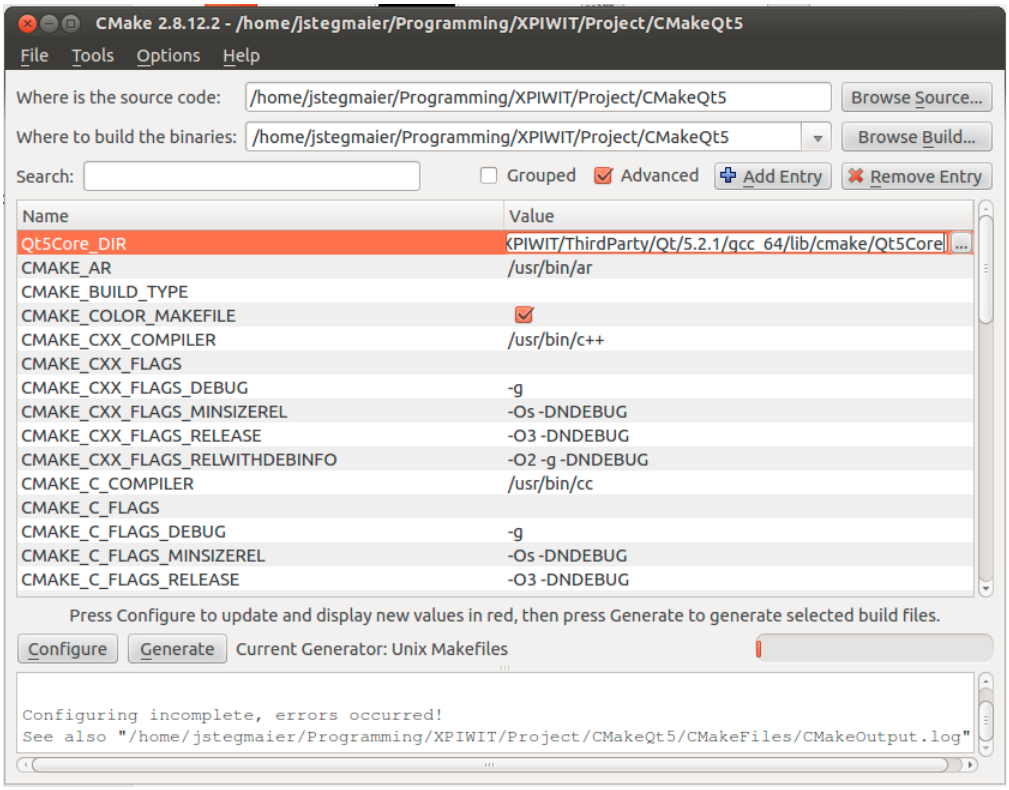

Figure 18: CMake settings if Qt path is not found automatically.

6 Development

6.1 Project Structure

XPIWIT uses CMake to realize platform independent project files. The project source code is structured into a core part, handling the input arguments and XML parsing and a filter part for creating and handling the images and filters. The core part is subdivided in the ”Main” part which handles the main event loop and the logger singleton object, the ”CMD” part handling the command line arguments and the ”XML” part handling both the parsing of XML pipelines and the creation of the filter list XML. The ”Filter” part is subdivided in five parts. The first part ”Base” contains the base class for the filter and the image as well as the metadata management. The ”ITKCustom” part contains custom ITK filter. The ”ThirdParty” part contains third party ITK filters. The ”XpWrapper” part contains wrappers for ITK filters for the use in XPIWIT and the ”XpExtended” part contains XPIWIT filters with custom filters or multiple ITK filter wrappers.

6.2 Wrapping ITK Filters for XPIWIT

Usually, the most convenient way to add a new filter wrapper to XPIWIT is to simply use our ”FilterGenerator.m” MATLAB script located in the folder ”Source/Template/”. The script allows to adjust properties like name, parameters, inputs and outputs of the filter and generates a template that only requires a few lines of additional code to add a new filter to XPIWIT. Alternatively, you can also start off by copying a similarly structured filter and adapt only the necessary locations. The following section describes the individual components used by our XPIWIT wrapper. Every filter used in XPIWIT must inherit from ProcessObjectBase in the XPIWIT namespace and provide an update method, which does the processing. A typical header can be seen in Lst. 16. Every XPIWIT filter should be in the namespace ”XPIWITFilter”.

Listing 16: XPIWIT wrapper header

```
1  #ifndef DISCRETEGAUSSIANIMAGEFILTERWRAPPER_H
2  #define DISCRETEGAUSSIANIMAGEFILTERWRAPPER_H
3
4  // namespace header
5  #include "../../Base/Management/ProcessObjectBase.h"
6  #include "../../Base/Management/ProcessObjectProxy.h"
7
8  namespace XPIWIT
9  {
10
11 template< class TInputImage >
12 class DiscreteGaussianImageFilterWrapper : public ProcessObjectBase
```

```
13 {
14    public:
15
16       DiscreteGaussianImageFilterWrapper();
17       virtual ~DiscreteGaussianImageFilterWrapper();
18       void Update();
19
20    static QString GetName() { return "DiscreteGaussianImageFilter"; }
21    static QString GetType() { return (typeid(float) == typeid(typename
          TInputImage::PixelType)) ? "float" : "ushort"; }
22    static int GetDimension() { return TInputImage::ImageDimension; }
23 };
24
25 } // namespace XPIWIT
26
27 #endif // DISCRETEGAUSSIANIMAGEFILTERWRAPPER_H
```

The static GetName method in the header specifies the name of the filter that will be used to reference the filter in the XML. The constructor defines inputs and parameter of every filter. An example is shown in Lst. 17. First of all the two member variables ”mName” and ”mDescription” have to be set. The member ”mName” is set to the name specified in the header and will appear in the filter list XML. The member ”mDescription” contains a short description of what the filter does. The second thing is to set up ”mObjectType” (Lines 9-20). With ”SetFilterType()”, the filter type can be set (mostly ProcessObjectType::FILTERTYPE_FILTER). With ”SetNumberTypes()” the number of template types has to be set (mostly 1). Now the in- and outputs has to be defined. With ”SetNumber*()” the number of image and meta in- and outputs can be defined. For every image one ”AppendImage*Type()” has to be called. If the image is processed in the first template type append 1 for the second template type append 2 and so on. For metadata append the name of the type (e.g. regionprops) with ”AppendMeta*Type()”. These information are important for the creation of the filter list. The third part is the definition of the filter parameter and their type(Lines 23-28). This is done by add the settings to ”mProcessObjectSettings”. The ”AddSetting()” method needs four inputs. The first input of type QString is the name of the parameter, the second input of type QString is the default value, the third input is the data type of type ”ProcessObjectSetting ::SettingValueType” (string, double, int, boolean) and the fourth input is the description of this parameter. To initialize the filter call ”ProcessObjectBase::Init()” of the base class.

Listing 17: XPIWIT wrapper constructor

```
1 template< class TInputImage >
2 DiscreteGaussianImageFilterWrapper<TInputImage>::
     DiscreteGaussianImageFilterWrapper() : ProcessObjectBase()
3 {
4   // set the filter name
5   this->mName = DiscreteGaussianImageFilterWidget<TInputImage>::GetName();
6   this->mDescription = "Gaussian Smoothing Filter. ";
```

```
7   this->mDescription += "Filters the image with a gaussian kernel defined by
        sigma.";
8
9   // set the filter type and I/O settings
10  this->mObjectType->SetFilterType(ProcessObjectType::FILTERTYPE_FILTER);
11  this->mObjectType->SetNumberTypes(1);
12
13  this->mObjectType->SetNumberImageInputs(1);
14  this->mObjectType->AppendImageInputType(1);
15
16  this->mObjectType->SetNumberImageOutputs(1);
17  this->mObjectType->AppendImageOutputType(1);
18
19  this->mObjectType->SetNumberMetaInputs(0);
20  this->mObjectType->SetNumberMetaOutputs(0);
21
22  // add settings
23  ProcessObjectSettings* processObjectSettings = this->mProcessObjectSettings;
24  processObjectSettings->AddSetting( "Variance", "1.0",
25    ProcessObjectSetting::SETTINGVALUETYPE_DOUBLE,
26    "Variance of the gaussian kernel." );
27  processObjectSettings->AddSetting( "MaximumError", "0.01",
28    ProcessObjectSetting::SETTINGVALUETYPE_DOUBLE,
29    "Maximum error of the gaussian function approximation." );
30  processObjectSettings->AddSetting( "MaximumKernelWidth", "32",
31    ProcessObjectSetting::SETTINGVALUETYPE_INT,
32    "Maximum kernel size in pixel." );
33  processObjectSettings->AddSetting( "UseImageSpacing", "1",
34    ProcessObjectSetting::SETTINGVALUETYPE_BOOLEAN,
35    "Use the real spacing for the gaussian kernel creation." );
36
37  // initialize the widget
38  ProcessObjectBase::Init();
39 }
```

The ”update()” method is called to process the filter. An example is shown in Lst. 18. The methods starts with the call of ”PrepareInputs()” from the base class to initialize the meta inputs. The second step is the parsing of the inputs parameter from ”mProcessObjectSettings” (Lines 6-17). The third step is to set the image pointer in the given template type. The method ”GetImage()” of the ImageWrapper class will cast and rescale the image if necessary and always return the image in the given type. Now the processing begins with a call of ”StartTimer()” of the base class (line 22). In the lines 24 to 36 the call of the ITK filter is done. Please check the ITK documentation for further information about available ITK filters. To store the image and return it back to XPIWIT a ”ImageWrapper” object has to be created and the image has to be set. The ”ImageWrapper” can handle any common image type and provide information about the dimensionality and type. The pointer has to be added to the ”mOutputImage” member. To stop the timer and log the performance

”LogPerformance()” of the base class has to be called. The ”Update()” method of the base class will handle meta-objects and write the results to disk if the ”WriteResult” flag is set. Finally, explicit template instantiations of the current filter automatically add the filter to the process object manager, making the filter accessible from the XML.

Listing 18: XPIWIT wrapper update

```
1  template< class TInputImage >
2  void DiscreteGaussianImageFilterWrapper<TInputImage>::Update()
3  {
4    ProcessObjectBase::PrepareInputs();
5
6    // get parameters
7    ProcessObjectSettings* objectSettings = this->mProcessObjectSettings;
8    const int maxThreads =
9      objectSettings->GetSettingValue( "MaxThreads" ).toInt();
10   const int variance =
11     objectSettings->GetSettingValue( "Variance" ).toDouble();
12   const double maximumError =
13     objectSettings->GetSettingValue( "MaximumError" ).toDouble();
14   const int maximumKernelWith =
15     objectSettings->GetSettingValue( "MaximumKernelWidth" ).toInt();
16   const int imageSpacing =
17     objectSettings->GetSettingValue( "UseImageSpacing" ).toInt()
18
19   // get images
20   TInputImage::Pointer inputImage =
21     mInputImages.at(0)->GetImage<TInputImage>();
22   ProcessObjectBase::StartTimer();
23
24   // setup the gaussian filter
25   typedef itk::DiscreteGaussianImageFilter<TInputImage, TInputImage>
26     GaussianFilter;
27   typename GaussianFilter::Pointer filter = GaussianFilter::New();
28   filter->SetInput( inputImage );
29   filter->SetReleaseDataFlag( true );
30   filter->SetVariance( variance );
31   filter->SetMaximumError( maximumError );
32   filter->SetMaximumKernelWidth( maximumKernelWith );
33   filter->SetUseImageSpacing( imageSpacing );
34   filter->SetNumberOfThreads( maxThreads );
35
36   itkTryCatch(filter->Update(),
37     "Error: DiscreteGaussianImageFilterWrapper Update Function.");
38
39   ImageWrapper *outputImage = new ImageWrapper();
40   outputImage->SetImage<TInputImage>( filter->GetOutput() );
41   mOutputImages.append( outputImage );
42
```

```
43   // log performance and write results
44   ProcessObjectBase::LogPerformance();
45   ProcessObjectBase::Update();
46 }
47
48 // explicit template instantiations to register the filter in the factory
49 static ProcessObjectProxy< DiscreteGaussianImageFilterWrapperWidget<
       Image2Float> > DiscreteGaussianImageFilterWrapperWidgetImage2Float;
50 static ProcessObjectProxy< DiscreteGaussianImageFilterWrapperWidget<
       Image3Float> > DiscreteGaussianImageFilterWrapperWidgetImage3Float;
51 static ProcessObjectProxy< DiscreteGaussianImageFilterWrapperidget<
       Image2UShort> > DiscreteGaussianImageFilterWrapperWidgetImage2UShort;
52 static ProcessObjectProxy< DiscreteGaussianImageFilterWrapperWidget<
       Image3UShort> > DiscreteGaussianImageFilterWrapperWidgetImage3UShort;
```

After a complete rebuild of XPIWIT, the additional filter should be usable in an XML pipeline. To check if the filter registration was successful, regenerate the overview filterlist as described in the quick start guide and check if your new filter is contained. In case the filter is not listed in the GUI yet, simply delete the ”myfilters.xml” that resides next to the GUI executable and execute the GUI again. The filter should subsequently be accessible in the GUI.

7 Gait-CAD Interface

XPIWIT can be used with the images and videos (IMVid) extension of the open source Matlab toolbox Gait-CAD[8] (GNU-GPL license, [10]).

7.1 Installation

The release of XPIWIT includes the folder Gait-CAD which contains Matlab files to create the filter plugins and callbacks. To generate the files use the ”setup.m” file. In this file a path can be specified to store the generated files to a temporary folder and copy them manually with the file ”callback_xpiwit_processing.m” file to ”\application_specials\imvid\plugins\mgenerierung\.”

For this step the filter list has to be created this can be done by a XPIWIT call with the parameter --filterlist $<$ path $>$. For example:

Listing 19: Create the filter list

```
1 D:\XPIWIT\xpiwit.exe --filterlist "D:/tmp/filterlist.xml"
```

Gait-CAD expects the ”XPIWIT.exe” in the following directory ”\application_specials\imvid \plugins\XPIWIT\”. The last folder has to be created if it does not exist. With these two steps Gait-CAD is ready to use XPIWIT and the filter should be automatically added to the plugin sequence list within the ”IMVid” plugin.

7.2 Use XPIWIT with Gait-CAD

The first step to use XPIWIT is the activation of the IMVid plugin if not already activated. This can be done within the menu `Extras  Choose application-specific extension packages`, select IMVid and activate it. A screenshot of Gait-CAD is shown in Fig. 19. The numbers in brackets of the following text refer to the number in the figure. To use the XPIWIT filters a project with images has to be loaded or create a new one by clicking on ”Images and Videos” (1) $\rightarrow$ ”import images” and select a folder with some images. Select the `Plugin sequence` in the drop down menu (2) to get to the shown environment in the figure. In the default case all available filters are shown on the ”plugin list” (4). With the drop down menu (3) it is possible to select only the ITK filter of XPIWIT. The parameters of an added filter can be defined with the drop down menu (6) and the textbox (5) The first parameter of every XPIWIT filter is the filter id, the second is the input image list and the third the meta input list. The following parameters are

[8] `http://sourceforge.net/projects/gait-cad/`

filter dependent. If the first three parameters are empty, Gait-CAD assumes that the pipeline should be processed linear and every filter has one input image and one output image. This behavior is forced if the checkbox (8) is activated. The advantage of this mode is the compatibility with the other Gait-CAD filters. The disadvantage is that every filter is processed separately by XPIWIT.

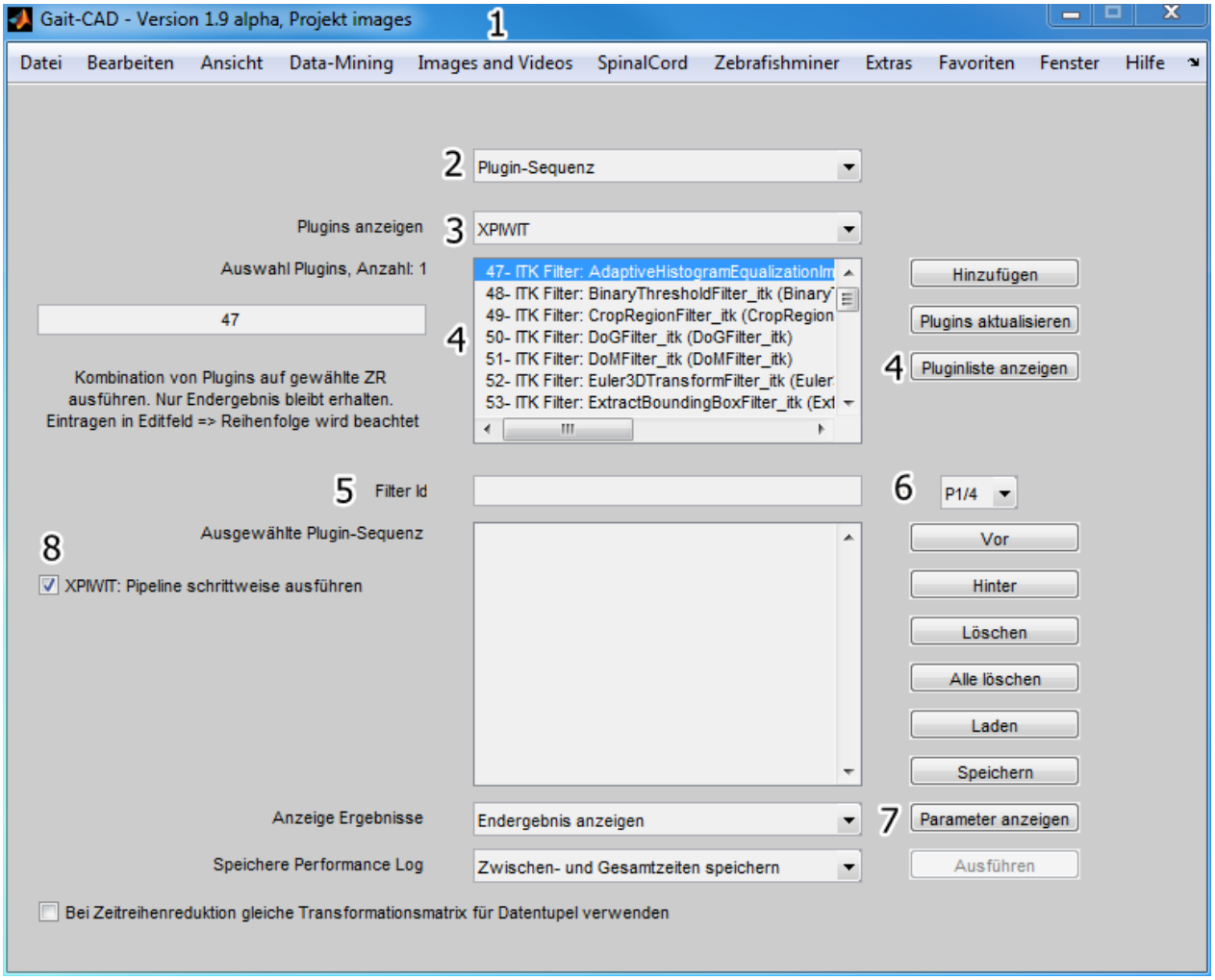

Figure 19: Gait-CAD screenshot.

If more complex pipelines should be processed the checkbox (8) has to be disabled. This means that Gait-CAD tries to put all filters into one pipeline. This mode has the ability to be faster, because images are handled by XPIWIT during the pipeline execution and not by Gait-CAD, but therefore the id's and input lists have to be set, if they do not fit the default case. The default case means that every filter takes the first image of the prior filter. In a case where a filter has two or more outputs the id has to be specified and the input lists of the filter that need this input has to be set. It is fine

for a filter to have only a specified id or input list. If the other parameter fits to the default case it will be included by Gait-CAD automatically. To get an overview of all parameter of the current pipeline the Button at (7) can be clicked to produce a file with all settings of the current sequence. Any changes in this file have no effect on the actual settings, so it can be handled as read only.

7.3 Handling of Complex Pipelines

For the creation of linear pipelines Gait-CAD is able to handle the pipeline automatically and no id or input has to be set. Complex pipelines, however, have to make use of this feature to work properly. Therefore it is necessary to explain some background. Every pipeline (also in stepwise mode) starts with an image reader with id `item_0001` this reader is automatically added to every pipeline. To get its output the following filter can refer to the id `item_0001` with its output `1`. This can be added as the second parameter to the filter. The id and the number are separated with ”,”(comma) if more than one input is specified they has to be separated with ”;”(semicolon). So if parameter one and two of the first filter are empty, Gait-CAD will implicit fill them with the id `item_0002` (consecutive numbering) and input `item_0001,1`. The next filter will get the id `item_0003` and the input `item_0002,1` and so on. The id as well as the input can be set by the user to create complex pipelines and to use filter with more than one input. It is possible to set any input to more than one filter and create parallel filter chains and to handle meta information.

References

[1] Stegmaier J, Otte JC, Kobitski A, Bartschat A, Garcia A, et al. (2014) Fast segmentation of stained nuclei in terabyte-scale, time resolved 3D microscopy image stacks. PLoS ONE 9: e90036.

[2] Tomer R, Khairy K, Amat F, Keller P (2012) Quantitative high-speed imaging of entire developing embryos with simultaneous multiview light-sheet microscopy. Nature Methods : 755–763.

[3] Mikut R, Dickmeis T, Driever W, Geurts P, Hamprecht F, et al. (2013) Automated processing of zebrafish imaging data - a survey. Zebrafish 10: 401-421.

[4] Stegmaier J, Mikut R, Gehrig J, Liebel U (2014) Analyzing multidimensional image data of live embryos using the Acquifer HIVE technology. Technical report, Karlsruhe Institute of Technology and Acquifer AG.

[5] Schindelin J, Arganda-Carreras I, Frise E, Kaynig V, Longair M, et al. (2012) Fiji: An open-source platform for biological-image analysis. Nature Methods 9: 676–682.

[6] de Chaumont F, Dallongeville S, Chenouard N, Hervé N, Pop S, et al. (2012) Icy: An open bioimage informatics platform for extended reproducible research. Nature Methods 9: 690–696.

[7] Kankaanpää P, Paavolainen L, Tiitta S, Karjalainen M, Päivärinne J, et al. (2012) Bioimagexd: An open, general-purpose and high-throughput image-processing platform. Nature Methods 9: 683–689.

[8] Peng H, Bria A, Zhou Z, Iannello G, Long F (2014) Extensible visualization and analysis for multidimensional images using vaa3d. Nature Protocols 9: 193–208.

[9] Cedilnik A, Geveci B, Moreland K, Ahrens J, Favre J (2006) Remote large data visualization in the paraview framework. In: Proceedings of the 6th Eurographics conference on Parallel Graphics and Visualization. Eurographics Association, pp. 163–170.

[10] Stegmaier J, Alshut R, Reischl M, Mikut R (2012) Information fusion of image analysis, video object tracking, and data mining of biological images using the open source MATLAB toolbox Gait-CAD. Biomedizinische Technik (Biomedical Engineering) 57 (S1): 458-461.