Universiteit Leiden
The Netherlands

Bioinformatics

High throughput acquisition of zebrafish larvae using VAST

Jurian Onderwater,

s2649438

Supervisors:
Fons Verbeek & Kristian Rietveld

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl 14/03/2023

Abstract

This thesis focuses on automating the high throughout acquisition of zebrafish larva images using the VAST and an external microscope, with the goal to generate high quality 3D models of those larvae. To achieve this, MicroManager was used, an open source microscope control platform, and the corresponding Python library to control a Leica DM6000B microscope such as changing zoom levels, light intensity, and automaticaly snapping images. These images are then uploaded to the LLSC computer cluster for further processing. All of this happens through a single push of a button in a GUI that was created using the QT framework for Python.

Contents

1 Introduction

1.1 High-throughput experiments

High-throughput experiments are very common nowadays in biology. The term high-throughput, in short, means the use of a lot of (computer)resources to process large amounts of (biological)data. In the case of this project, the high-throughput part of the experiments refers to the large amount of necessary computer resources to keep up the captured images by the microscope: per zebrafish subject between 240-300. All of these images are then sent on to a server for post-processing and to be made into a 3D reconstruction.

1.1.1 3D zebrafish models processing pipeline

The goal of this project is to make the high-throughput acquisition of zebrafish larvae images easier and faster. The images of the zebrafish larvae are used for creating 3D models of said larvae. This process contains a whole procesing pipeline (fig 1), of which this project will span the first steps. A computer program will be created that interacts with the existing acquisition hardware including but not limited to a Leica microscope and the VAST (Elaboration on this hardware follows in sections 2.1 and 2.5). Then another computer program will integrate the automated high-throughput acquisition with the rest of the processing pipeline, by uploading the images to the LLSC server where a computer cluster will take care of the postprocessing and the 3D reconstruction and the 3D reconstruction is performed.

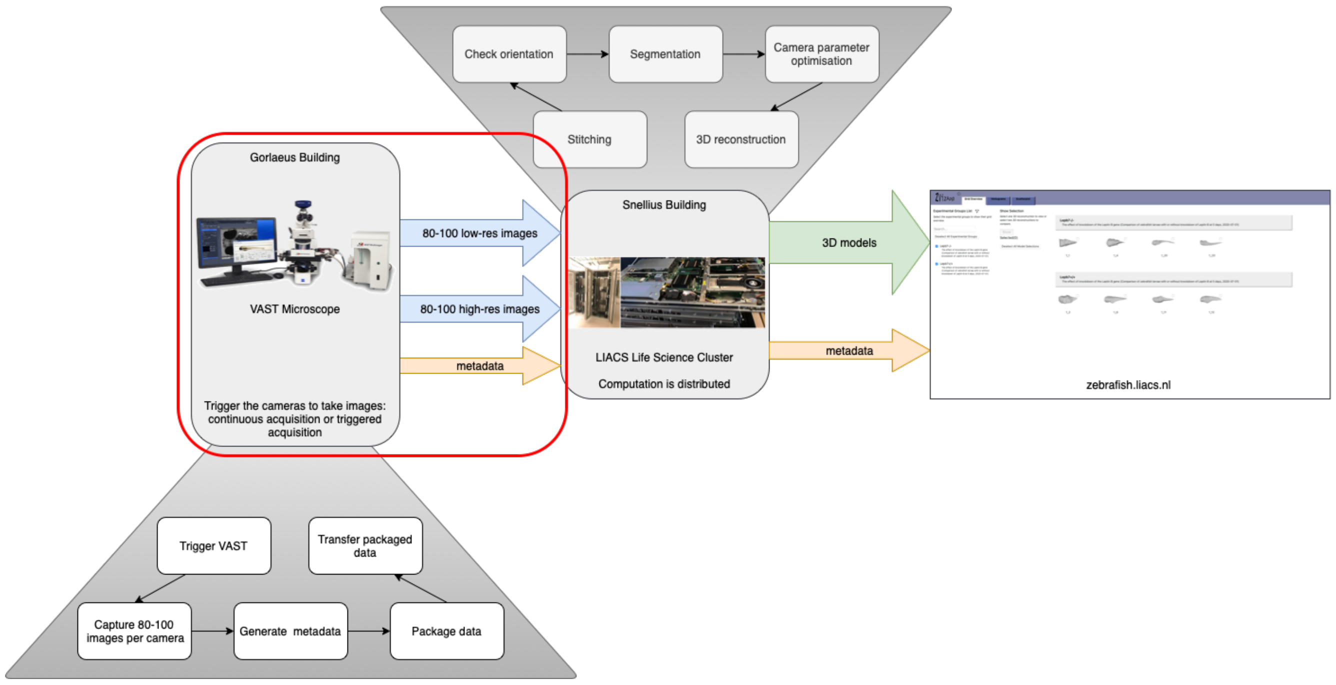


Figure 1: The scope of this thesis, circled in red. For a higher resolution version of this image see Chapter 7

1.1.2 VAST

The VAST (fig 2) is a machine that can handle a lot of the zebrafish handling automatically, greatly reducing the amount of manual work and time that a researcher needs to spend on getting zebrafish in place for screening. The larvae are placed into a thin tube. In this tube they can be moved around and rotated by a series of stepper motors that move the tube. All of this happens while taking pictures of the larva. A more detailed description of the VAST follow in section 2.1

1.2 Previous work

The idea that we can use the VAST microscope for high-throughput imaging and making 3D reconstructions is a project that has been going on at Leiden University for a while. In 2017, Yuanhao Guo et al. proposed a method for using the VAST and other microscopes to make 3D reconstructions of zebrafish larvae, in [2]. Since then there have been improvements to the whole workflow such as in 2018, [3] proposed a method of robustly segmenting the larvae from the background of the images to make better 3D models.
Other projects have also worked towards improving this workflow. For example: Rosa Zwart, who made the visualisation website[12] and Josef Hamelink, who worked on optimisation of 3D reconstruction parameters[4].

1.3 Research Question

The goal for this thesis is to improve how the acquisition part of this whole workflow is handled, by finding out how this can be done in a fully automated manner. The state of this process at the beginning of the thesis is that all images of a single larva have to be manually captured and sent over to the computer cluster where the reconstruction happens. My goal is to create a system that makes it possible that the images are all captured automatically with the push of a single button and when this is done, they should also be automatically uploaded to the right place for processing and 3D reconstruction.

1.4 Thesis overview

This thesis will first go over the materials and methods used for answering the research question, in chapter 2. The next chapter will then elaborate on how this all was used to solve the problem in the section Design and Implementation. Then it will talk about testing this in the Experiments chapter. Lastly, an attempt will be made to draw conclusions from what was learned here, and the results will be discussed in the Conclusions chapter. The conclusions chapter will also go over some discussion: what went wrong or does not work yet.

2 Materials and Methods

2.1 VAST

VAST (Fig. 2)is a microscopy machine made by Union Biometrica. The name VAST stands for *Vertebrate Automated Screening Technology*. It was created as a machine to speed up the screening of zebrafish larvae, by automating the handling of the larvae prior to - and during the screening. This includes the loading of the larvae from a storage container into a sample cup, from where they can transported one by one into the capillary where the imaging takes place, but also figuring out the rotation of the fish and rotating it to the desired angle, and then when the imaging is done, the larva is automatically sent on to either a waste container or another place for further storage (Fig. 3).
VAST can also be mounted on other microscopes, to greatly expand it's imaging capabilities. For this project a Leica microscope was attached (sec. 2.5). This microscope is then used to capture higher quality data.

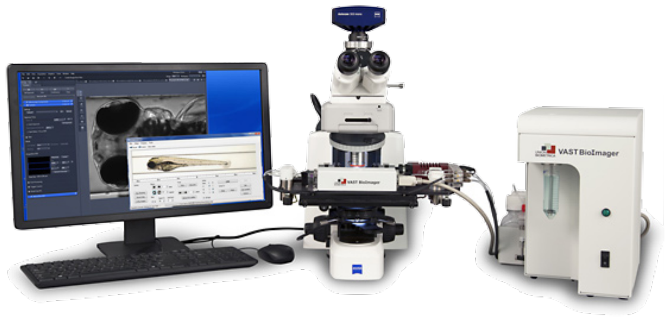


Figure 2: The VAST microscope system

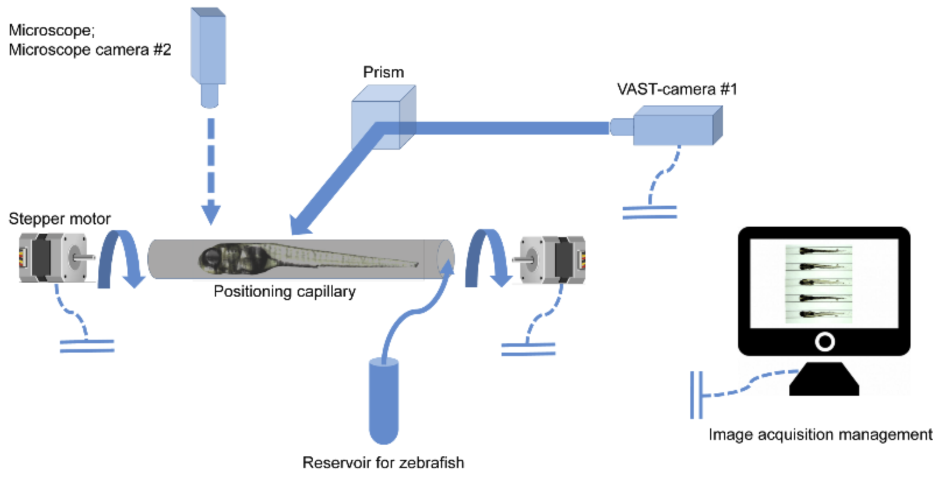


Figure 3: The VAST transports zebrafish larvae from a storage container (Reservoir for zebrafish) into a capillary (positioning capillary), where it can rotate and move them to capture images from the desired perspective using a stepper motor[2].

2.2 LLSC

All of the heavy computation that needs to happen to construct 3D models of the captured zebrafish larvae, does not happen locally on the computer used to carry out the acquisition. Instead, this happens on the LLSC cluster. LLSC stands for LIACS Life Sciences Cluster. This is a small computer cluster dedicated to life sciences research, located in the Snellius building at the Leiden Bio Science Park. On this cluster all of the segmentation of the larvae from the background is carried out, as well as the optimisation of the reconstruction parameters, and the actual 3D reconstruction. For connecting the VAST to the LLSC, a program was written that polls a folder for new items and sends them over to a server connected to the LLSC.

2.3 Connection between VAST and PC

The VAST was not originally intended to be used for creating 3D reconstructions and high-throughput acquisition. Yet, with some modifications and external hardware we are able to automate it's usage for fast creation of 3D reconstructions. An Arduino micro-controller is used to send and receive signals from the VAST over a USB serial connection, which can be used to control the VAST. Figure 4 and table 1 contain a diagram and the commands used in the connection between the VAST and the Arduino.

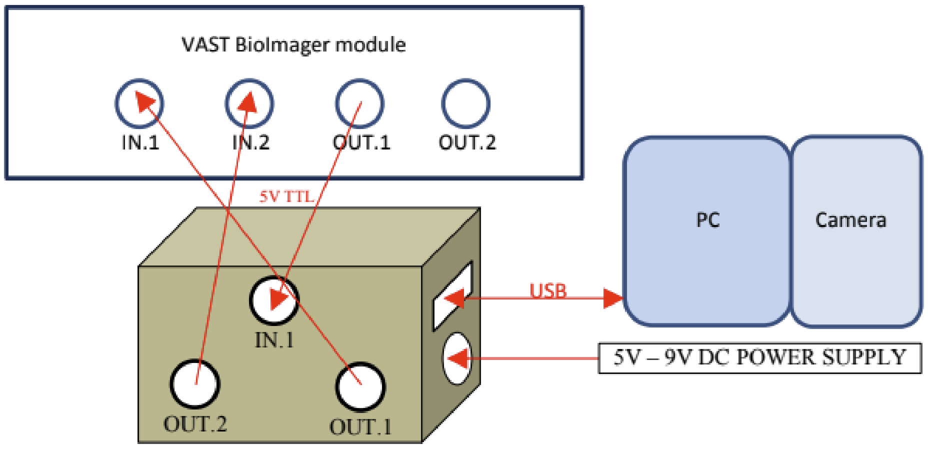


Figure 4: Diagram depicting how the VAST, Arduino, and PC are connected. The VAST BioImager module is used to control the functionality of the VAST using commands that the arduino can send over it's connection to the VAST.

VAST	Arduino	Description	Serial bus command
In.1	digital i/o pin 4	proceed to next larva	TT
In.2	digital i/o pin 5	forward larva to output container, not to waste	TS
OUT.1	digital i/o pin 2	larva is in place, ready for external imaging	R1
OUT.2	digital i/o pin 3	not currently used	

Table 1: IO lines and corresponding commands. The serial commands are sent over the USB connection between the PC and the arduino.

2.4 Programming and development environment

For this project, Python was chosen as the preferred language of implementation, because the rest of the components of the 3d reconstruction processing pipeline are also written in Python, and it is will be the easiest to maintain and understand for others in the future.

2.4.1 Python

Python version 3.9.2 and 3.10 were used throughout the development process. The tested lower limit for the Python version is 3.8.x. Below 3.8.x, some dependencies do not work.

2.4.2 QT

For the GUI that goes along with this project, the QT framework was chosen. QT is a framework used to create user interfaces. While it is originally meant for C++, it also has Python support, which makes it perfect for this project. Another big reason why QT was chosen is that its code looks a lot cleaner, thus being easier to maintain, than competitors like PyGui and Kivy.
QT was used to create a GUI in which the user can either start the process of capturing images with the microscopes using the program written for this thesis to start the 3D reconstruction process.
For building the GUI, QT was used inside Visual Studio Code for the functionality and inner workings, together with the QT-Creator app, for fixing the exact layout and aesthetics since the QT-Creator has a GUI which can be used to create apps, instead of pure code.

2.4.3 Paramiko

For transferring the images to the server a Python library called Paramiko was used. This library allows you to make a connection to a server and transfer files using SFTP, SSH File Transfer Protocol. It can do this in parallel, so if there are multiple images that can be sent over, there is no need to wait for images that are not yet done uploading.

2.4.4 MicroManager & PycroManager

To connect to the Leica microscope/camera there were two different options: use the Leica/BaumerOptronic API, or use MicroManager. Micromanager is an open source software package that can be used to automate the use of a wide variety of microscope hardware. Technically this still uses the BaumerOptronic API to communicate with the Leica microscope cameras, but this way it will be easier to use because of the more readily available documentation.
For connection to the specific microscope a configuration file was used that was generated using MicroManager's hardware configuration wizard, that lets you select devices you plan to use and configure their settings.
For integrating MicroManager into the program PycroManager was used. This is a Python library that works through MicroManager to control the microscope hardware via Python. PycroManager makes a connection to MicroManager through a local server that can be set up through MicroManager, to connect to the Java back-end that MicroManager uses [9]. Through PycroManager we can control all the functions of the microscope that we can control through MicroManager.

2.5 Leica Microscope

As discussed before in chapter 1, besides just the VAST, a higher quality microscope/camera combination from Leica is also used in conjunction with the VAST camera to capture images of the zebrafish (Fig. 5). This combination gives us images that are of a higher quality than the VAST, resulting in more detailed 3D models.
The exact microscope used is the Leica DM6000B in combination with the CTR6000 electronics box which functions as a controller for the entire microscope. The controller is also connected to a computer through USB which makes it possible to use software on the computer to control the microscope's functions. While this is originally intended to use with Leica's own software, it is also possible to use MicroManager.

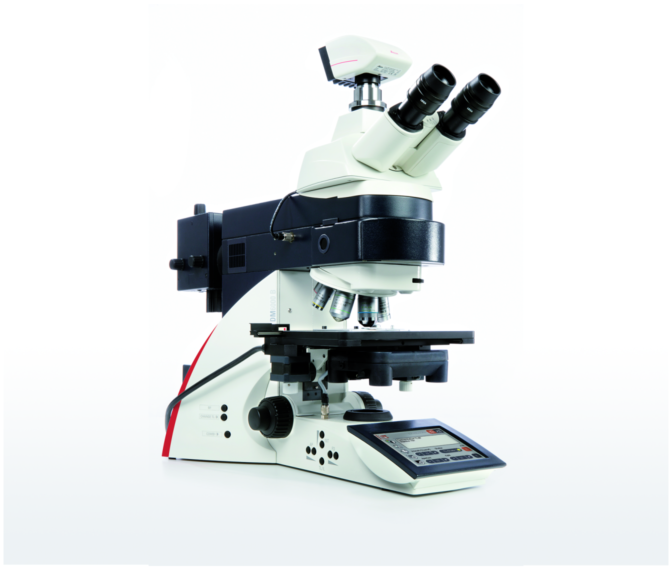


Figure 5: Leica DM6000B research microscope with the DFC450C camera attached [7]

With this microscope it is possible to perform both bright-field and fluorescence microscopy. The microscope is connected to a Leica DFC450C microscope camera. This DFC450C camera is controlled through the above mentioned Baumeroptronic API, in MicroManager.
While this microscope has the ability to move the subject on the X and Y axes, this has been set to a single stationary position, since the VAST can do all the necessary movement itself.

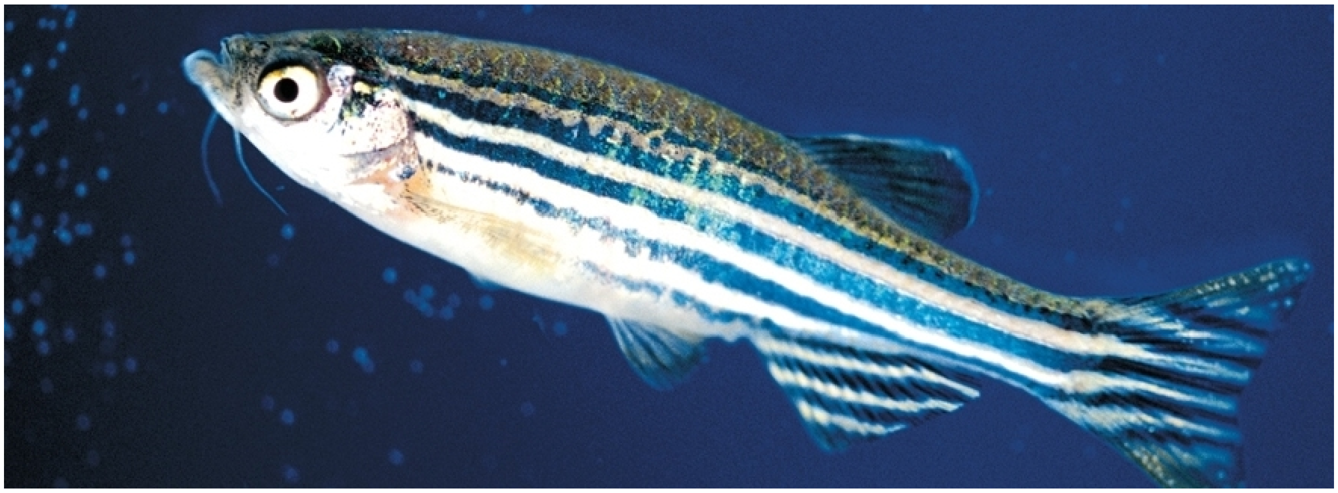
2.6 Zebrafish

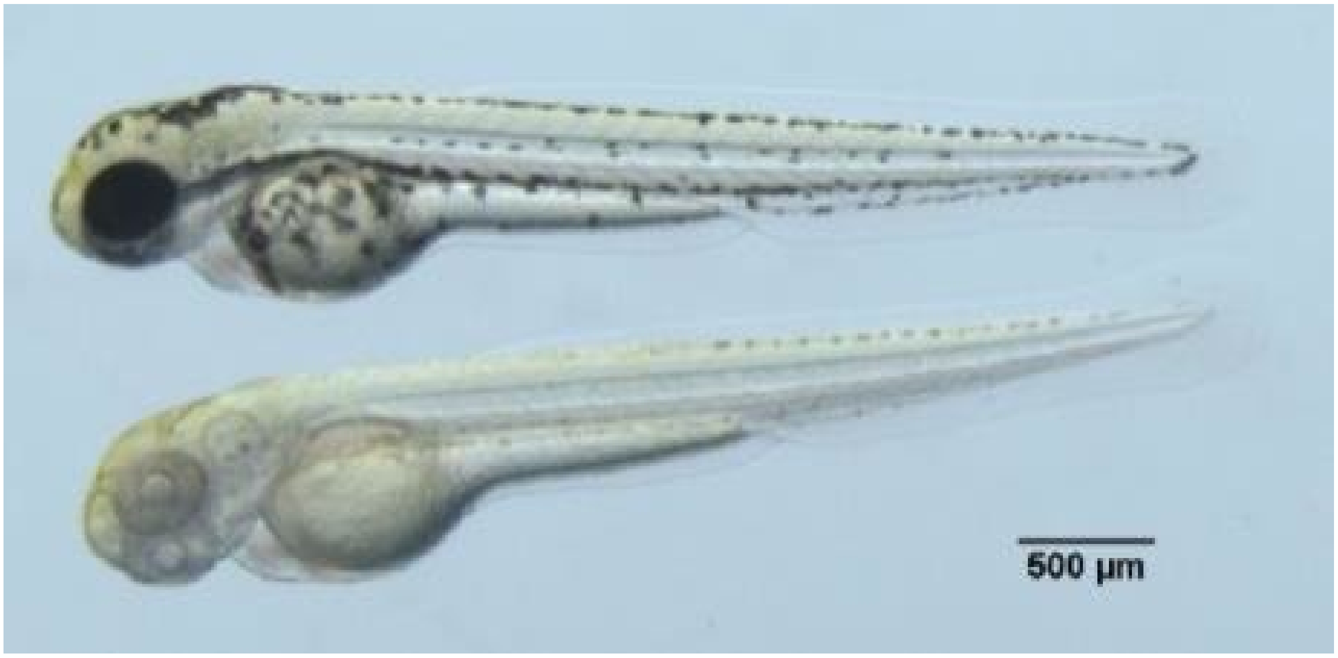
Zebrafish (Fig. 6a)are a commonly used organism for (high-throughput)biological research because of traits including, but not limited to:

- relatively high shared genome structure with humans [5][10].
- easily manipulated genetic material
- semi-transparent larvae (Fig. 6b).

- high reproductive rate and easily obtainable embryos [2].
- Their entire genome has been sequenced
- Fertilisation can happen externally [6]
- The larvae are nearly transparent

These traits allow for rapid experimentation, and automating the process of capturing the data makes all of this even faster. In addition to these traits, the number of embryos that can be obtained also make the zebrafish a good option for high-throughput experimentation.


(a) Zebrafish [11]



(b) Zebrafish larva [8]

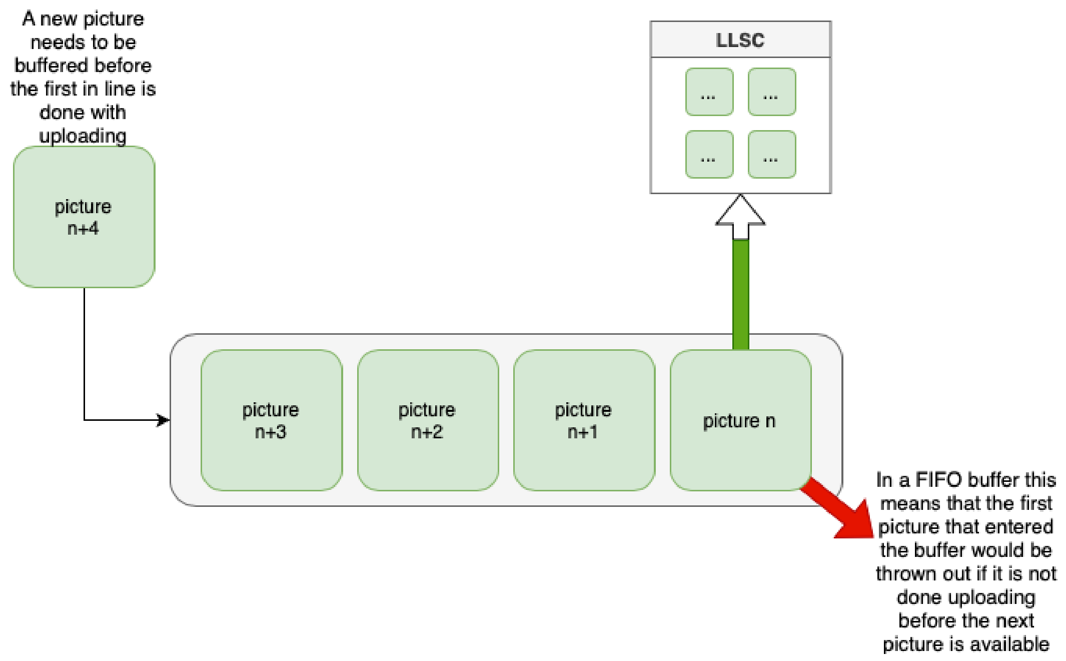
Figure 6: Zebrafish in mature - and larva form

3 Design and Implementation

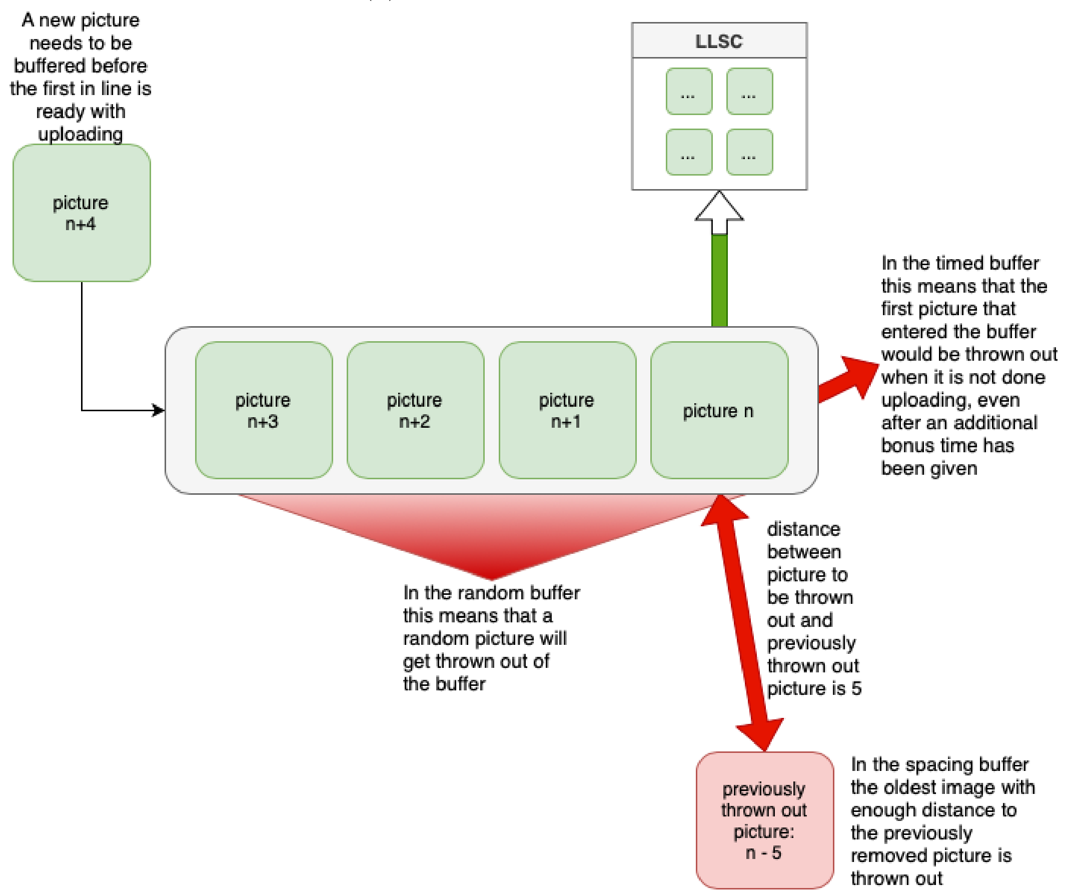
3.1 Data Transfer

As introduced in sections 2.2 and 2.4.3, a program was developed that scans a folder for new images and uploads those images to the LLSC using Paramiko, which supports parallel uploading.
This scanning and upload program first adds the new images it finds in the folder to a queue, implemented as a circular buffer, through one of various ways. That way there is always something to send over and the connection is never idling during the acquisition. When the buffer is full a new spot can be made available by either uploading an item to the LLSC or if an item is stuck, throwing that item out of the buffer. Uploading that item can be retried later.
Four different - though all based loosely on FIFO - buffering algorithms have been implemented, with visualisations in figure 7:

- Standard first-in-first-out (FIFO) buffer. The oldest item in the buffer gets removed if a space is needed before it is done uploading and uploading of that item gets retired at a later point.
- A timed buffer. This is the same as the FIFO algorithm, but with an added wait time to avoid throwing items out of the buffer that are likely almost done uploading.
- A spacing buffer. While the previous algorithms are based on existing algorithms, this one was specifically designed for our purpose of creating a 3D model in the end. Here, before an image is thrown out of the buffer, a check happens that checks when the last image was removed from the queue, and to not throw the current image out when the previous "deletion' happened only n pictures ago. This is meant to ensure that during reconstruction of the 3D model, flat spots do not happen when there are connection issues and multiple images in a row did not reach the server in time.
- Finally, a random buffer. If the buffer is full, in this case, a random item will be thrown out. This is not necessarily supposed as an actually used algorithm, but more to test the other algorithms against.
- Buffers for which the base code is complete but buffer specific changes are not implemented yet:
  - A second-chance buffer. This algorithm checks if an item in the buffer has been checked before, and only if the item has been seen in the buffer before, will it be thrown out. This buffer was not finished in time. Groundwork is all complete and implementation in the final product should not present major problems.
  - A least-recently-used (LRU) buffer. The LRU buffer checks which item in the buffer has started uploading the first. In normal conditions this is the same as FIFO. It differs when an item gets stuck in the uploading process and starts to hog space in the buffer.


A new picture needs to be buffered before the first in line is done with uploading
picture n+4
LLSC
...
...
...
...
picture n+3
picture n+2
picture n+1
picture n
In a FIFO buffer this means that the first picture that entered the buffer would be thrown out if it is not done uploading before the next picture is available

(a) FIFO buffer visualised


A new picture needs to be buffered before the first in line is ready with uploading
picture n+4
LLSC
...
...
...
...
In the timed buffer this means that the first picture that entered the buffer would be thrown out when it is not done uploading, even after an additional bonus time has been given
picture n+3
picture n+2
picture n+1
picture n
distance between picture to be thrown out and previously thrown out picture is 5
In the random buffer this means that a random picture will get thrown out of the buffer
previously thrown out picture: n - 5
In the spacing buffer the oldest image with enough distance to the previously removed picture is thrown out

(b) Buffers derived from FIFO, visualised

Figure 7: Uploading buffers visualised

3.2 Data Control

As the acquisition of data is only part of a larger pipeline, some metadata needs to be stored for later use during other parts of the process in order to keep a common data structure throughout the whole pipeline. For this purpose, the idea is using a JSON struct to store information on the acquisition, such as Rotation angle, amount of images captured, resolution. This will then be used in tasks such as optimisation of the reconstruction parameters, the visualisation of the models and their properties on the website, and other parts of the larger processing pipeline. Using a JSON struct was already proposed by Loes Dekker, for tracking the whole process [1], and Rosa Zwart for the website that displays the 3D reconstructions [12].

3.3 Data Acquisition

3.3.1 Standard

As the VAST does not work with MicroManager, it needs to be controlled in a different way. The capturing of images using the VAST camera happens through the already existing control sequence file for the VAST, to which I have made no changes. A file like this simply contains actions that the VAST has to execute, line by line.
The acquisition of images using the Leica microscope is also dependent on this control file: the delay between taking photos with the Leica microscope is dependent on how quickly the VAST takes an image, rotates, and takes the next image.

3.3.2 Leica: Brightfield & Fluorescence

The capturing of the images using the Leica microscope generally happens using the *multi_d_acquisition_events()* and *acquisition()* functions from Micromanager/pycromanager. In our case, the multiple dimensions used are the two channels: brightfield and fluorescence.
The *multi_d_acquisition_events()* function generates an event dictionary that contains properties of the images to be captured, like the time interval, which channel to use, or how many images in total there are to be captured. This dictionary is passed as an argument to the *acquire()* function which will then control the microscope and capture all the images.
The problem with this method, however, is that it does not always work. Even with no changes to the code, it happens every so often that *acquire()* simply refuses to produce images and just opens up the ndtiff viewer interface without showing anything. For this reason an additional method was created that mimics the functionality of an event dictionary and the acquire function. This is done using snapping single images with the built in *snap_image()* function, and putting that in a loop.

3.4 Post-Processing the data

After capturing an image with the microscope, it is still only an array of pixel-data. To make it usable for making a 3D reconstruction of the larva, some post-processing needs to happen to this array of pixels.

3.4.1 Saving individual images

When acquiring data through MicroManager/pycromanager, there is no built-in option to save every acquired image individually when using the *acquire()* function. Since saving the images individually instead of in one big multi-image file is necessary for uploading the images to the server in parallel, this functionality needs to be added. This was done using a custom image processing function. The *acquire()* function can take such a function as one of its arguments and use it to process all of the images it acquires. This behaviour will be used to intercept the raw pixel-data and save it as a PNG file and add the original metadata to that PNG file as well. The reason for not using the original TIFF file format is that those files are incredsibly large, which would make the uploading process a challenge. The image processing function then simply returns the unaltered pixel-data and metadata dictionary so the behaviour of the rest of the pipeline can continue as if nothing happened. This ensures that besides all the individual images that were acquired, the ndtiff file is also still generated in the end.
Because this method uses the acquire function, it does not work 100 percent of the time. To save individual images using the custom image taking function, the array of pixels is taken from the image that was just snapped using the *snap_image()* function, this array is reshaped from 1D into 3D (height of the image × width of the image × channels per pixel), and then the alpha channel is taken out of this array. The processed array is saved as a png file. The exact workings of this function are really similar to the what the *image_process_function()* mentioned above does. The only difference is in the way the pixel array is obtained. In *image_process_function()* this is automatically passed as an argument, while for the *snap_image()* function this is done by calling the pix method on the most recently snapped image which is automatically tagged by pycromanager.

3.5 Application interface (GUI)

Building an interface with QT works through adding windows to the application. The standard window is called the main window, which is what is used for our application. To this main window, widgets are added:

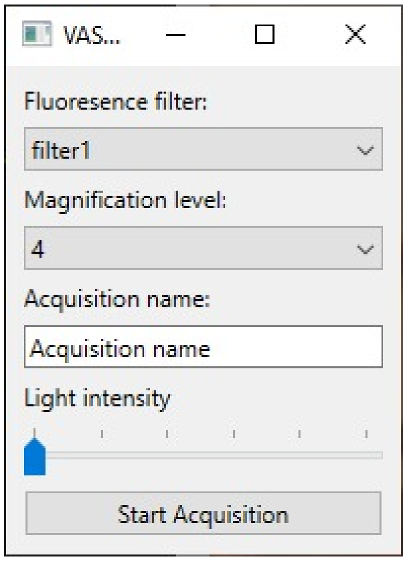


Figure 8: The GUI as rendered on a windows machine.

- Multichannel widget: a drop-down menu to select the desired filter for use with fluorescence microscopy.
- Magnification level widget: a drop-down menu to select the desired lens for capturing the images.
- Acquisition name widget: a text input box where you can specify what filename/path your acquisition should be saved as. When this is left blank, a standard name will be used by providing a keyword argument in the file-naming function.
- Light level widget: a slider with range 0-255 that controls the light level of the microscope
- Start widget: a button to start the acquisition and upload processes.

4 Results

This chapter will go over the flowschemes that resulted from implementing the software discussed in previous sections.

4.1 Data transfer

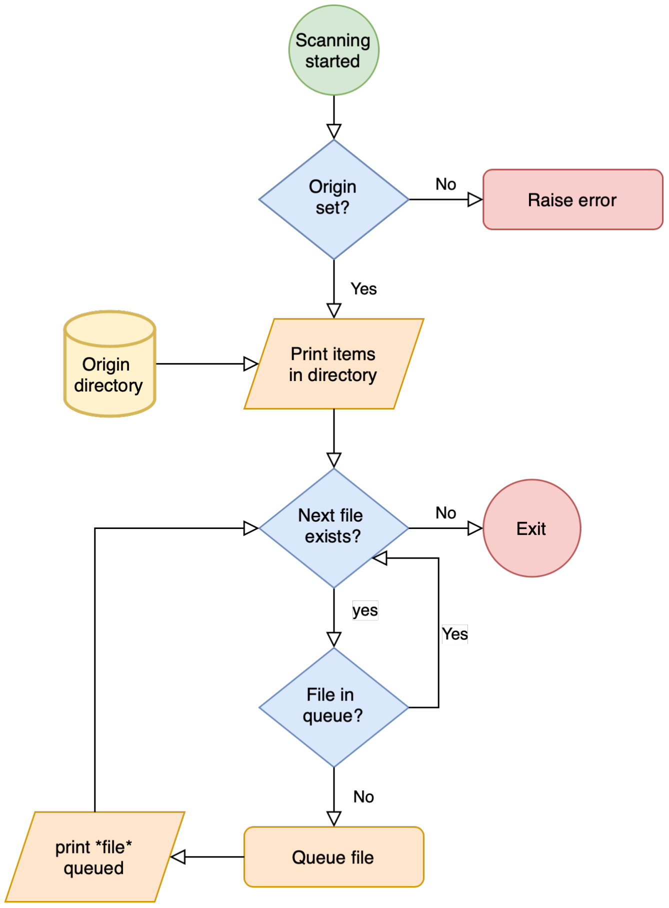


Figure 9: Flow scheme of the resulting scanning software.

The scanning process (Fig. 9) starts off by checking whether the user has provided the program with a location where it should scan -the origin folder - and throws an error if this is not the case. Then, for debuging purposes, all of the files in the folder are put in a list and printed in the terminal. Next, the scanner will start looping over the files in the list that was just made, queue them if they are not already in the queue, and report back through a print command in the terminal that it has queued a file, and move on to the next file in the list. After each iteration of the loop, the list with files is updated to include newly captured images. When the folder is completely emptied out and all files are uploaded, the scanner stops scanning.

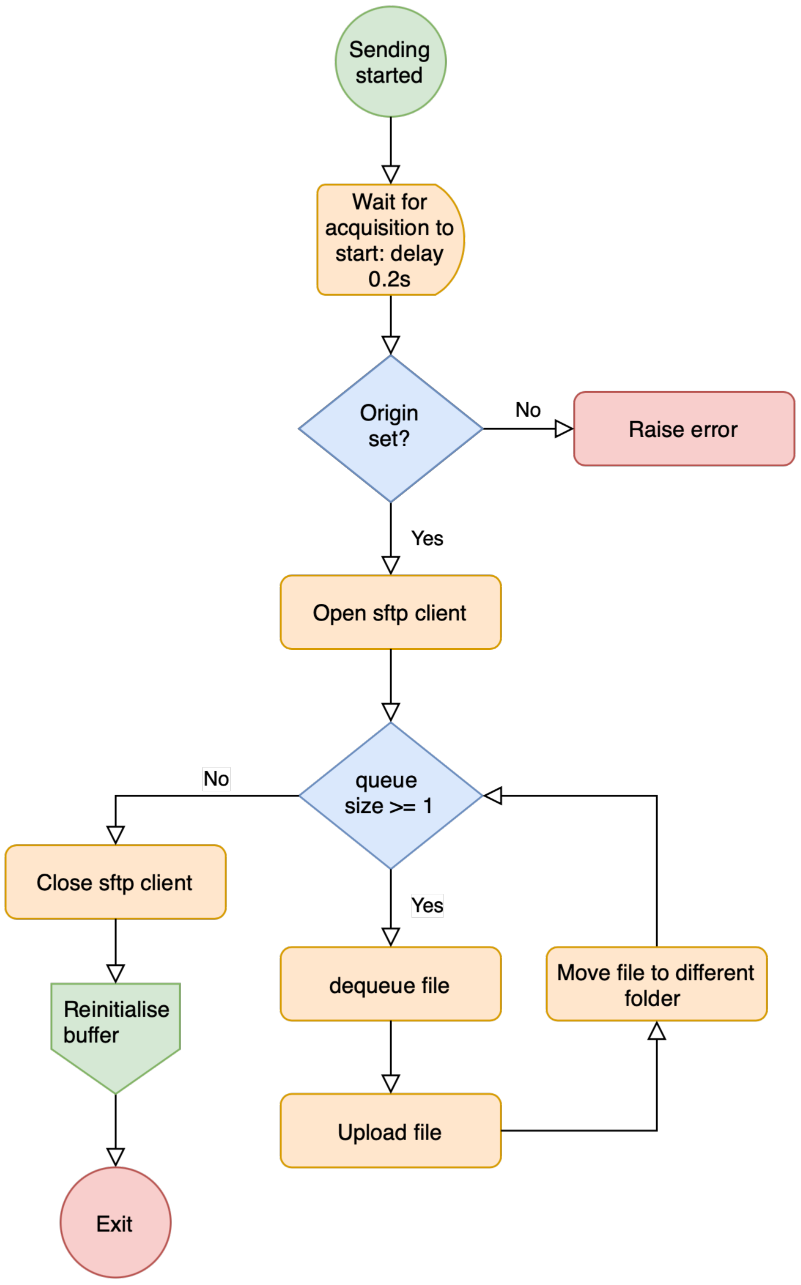


Figure 10: Flow scheme of the resulting transfer software.

The sending process (Fig. 10) also starts with checking if an origin folder has been provided, but first waits for 0.2 seconds to allow the acquisition to start up. Then, an sftp client is opened on the connection to the LLSC server. The sending process uses a while loop to check if the queue filled by the scanning process is still larger than 1, and empties it if it is. A file is dequeued, uploaded to the LLSC server, and moved to a different folder so the scanner does not add it to the queue again. After all images have been uploaded, the sftp client is closed, the buffer values reinitialised for the next acquisition, and the sender stops.

4.2 Connecting to LLSC

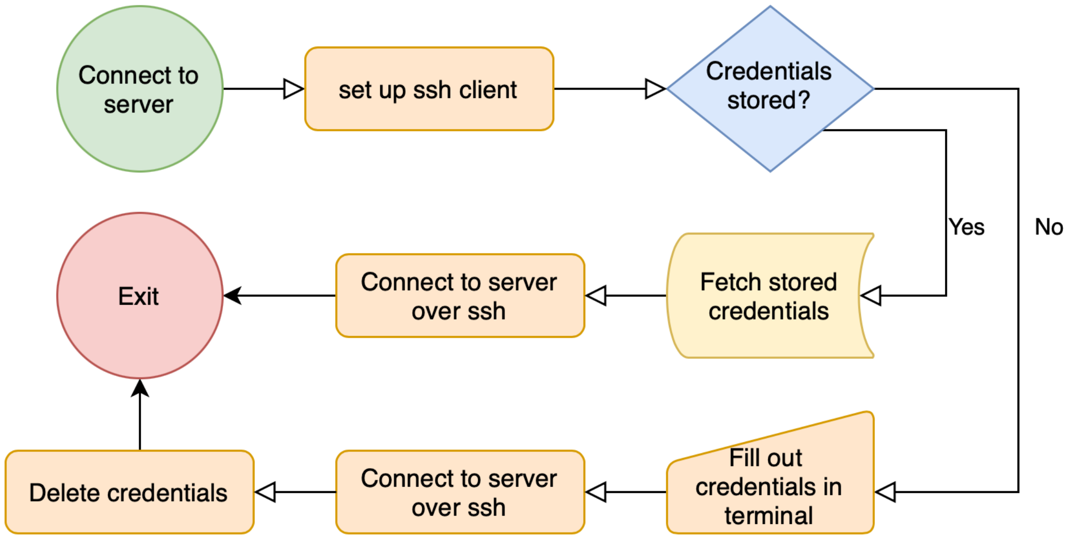


Figure 11: Flow scheme of the resulting server-connection software

Connecting to the LLSC server (Fig. 11) happens through ssh. When the connection is set up, the system first checks whether credentials are already stored. Usually this is the case, since we only connect to the LLSC server and do not need different credentials every time. If no credentials are stored, the system asks the user to fill out the hostname and their username and password in the terminal. The connection is then set up, and - in the case of the credentials being filled in in the terminal, they are deleted from memory again.

4.3 Data acquisition

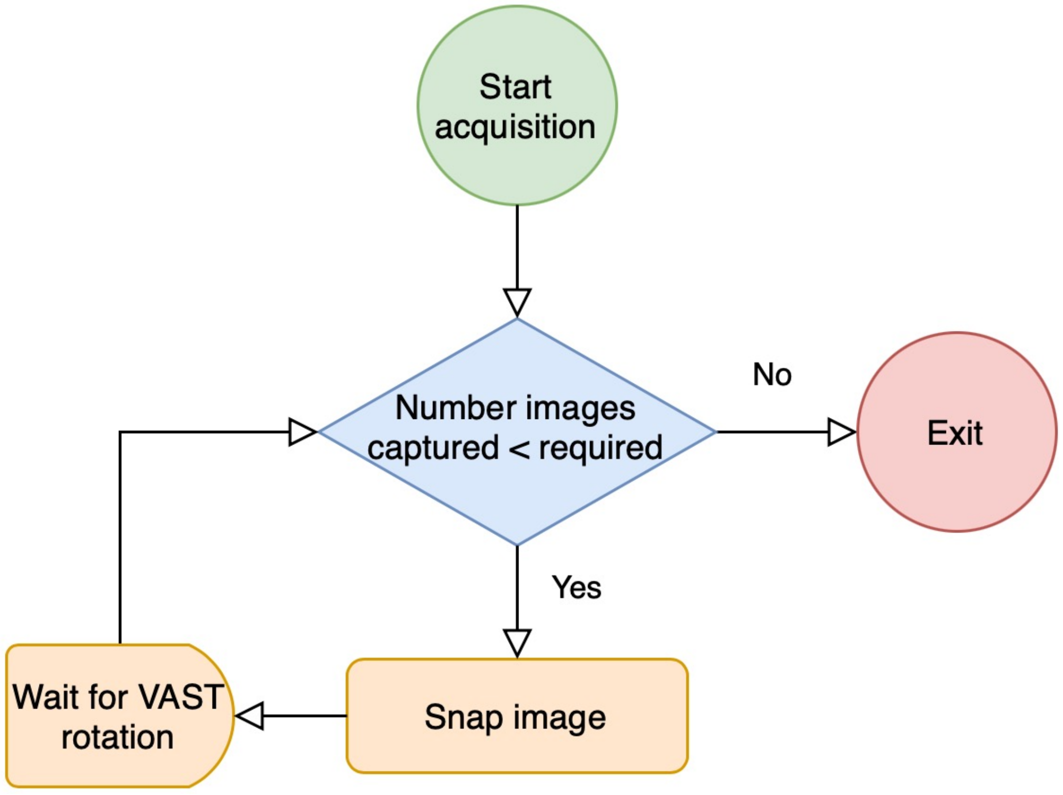


Figure 12: Flow scheme of the resulting image-acquisition software.

The acquisition process (Fig. 12) is made up of a for loop that captured images until it reaches captured photos. In every iteration, it also waits a small amount of time so that the VAST will have time to rotate the tube. After reaching 100 images, the process stops.

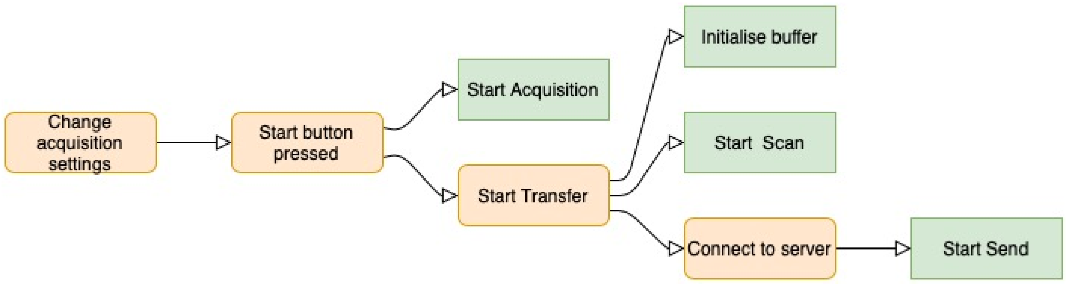


Figure 13: Flow scheme of the entire acquisition workflow in the resulting software.

Figure 13 contains the flow scheme of the entire workflow. First, the user can change the acquisition settings to their needs, after which they press the 'Start Acquisition' button. This button starts up two processes: Acquisition and Transfer. Transfer contains two sub processes: send and scan. So really all three processes are started when pressing the start button, with send being slightly delayed because there needs to be a connection to the server first.

5 Conclusions

5.1 General conclusions

The delivered software product is a significant step towards doing all of the acquisition automatically. Images do not have to be captured manually anymore, but can all be captured with one press of the start button in the GUI. Besides this, the user now also does not need to open all the different programs that were needed to do this job manually: VAST software, Leica's acquisition software, a program like Putty for transferring files. With doing this manually also comes the risk of human error, like forgetting to rotate the VAST, forgetting to take an image, misspelling a filename, and much more. This is all mitigated by using the automated system.
In a single sentence: By using the automated acquisition system the user can acquire the needed images faster, and with lower risk of user error, than when they would do the same acquisition manually. This all leads me to conclude that, apart from a couple of points discussed in section 5.2, I succeeded in reaching my goal set for this thesis in section 1.3.

5.2 Discussion

5.2.1 Connection of PC controls to the microscope

One big problem of the proposed work is that the connection between MicroManager/pycromanager and the microscope appears to be spotty at best. This can be concluded from the fact that after using the GUI to control the microscope, the microscope, or at least certain functions of the microscope, often becomes completely uncontrollable. The light can not be controlled anymore from either MicroManager, the GUI, or the buttons on the microscope itself. This also happens with the Z-stage, or focus drive, wheel. After using the GUI, the wheel on the microscope becomes unresponsive. The reason that I concluded that this is an issue in the connection between Micro - or PycroManager and the microscope, and not an issue in the code, is that the side effects of running the code can be observed in MicroManager's own property browser. When changing the light-intensity in the GUI and refreshing the property browser, it shows up changed to what the GUI just changed it to, without actually changing. This hints that the problem happens between the microscope and MicroManager, be it an issue with the cable, or something about the configuration file that causes the connection to fail, or something else.
Solving this issue did not fall under the scope of the project and MicroManager's maintainers have been made aware of the issue.

5.2.2 Blue hue in images

The images captured by the system right now all show a significant shift in their colours to blue, when the live feed in MicroManager shows the real image to be more orange. This could be an issue of colour space mismatch, or a bug in my code that has not yet found.

5.3 Future work

5.3.1 Different magnification factors

Future students continuing this project will have to implement a feature that deals with different magnification factors. When using higher magnification factors, it can happen that not the whole of the subject is visible in one frame. Because of this multiple images may need to be combined - one of the left side of the subject and one of the right side - into one image and stitch them together.

5.3.2 Connect VAST to software

Because of issues with the VAST, there also was no time to connect it to my software, so for now they run independently. Another addition that could be made is in the application, adding a number indicator that shows what light level is being used, since the range is so big you can not see it's exact value on the slider itself.

5.3.3 New ideas

For future work and improvements to the delivered system that were not already discussed, we could think about adding more controls to the user interface, such as control over the Z-stage, making it possible to get the subject in focus automatically.
Another valuable addition would be piece of code that can take commands from the user and generate new instructions for the VAST on the fly. This would greatly expand the versatility of the whole system by making it possible to change the amount of pictures that is taken per full rotation, the part of the subject that is in view, and more.
A future feature that might also be useful is also automating the loading of a new larva in the GUI, making the acquisition acquire images of multiple larvae instead of just one.

6 References

References

1. Dekker, L. Optimization of High-Throughput Zebrafish Imaging on a Distributed Computer (2019).
2. Guo, Y., Veneman, W. J., Spaink, H. P. & Verbeek, F. J. Three-dimensional reconstruction and measurements of zebrafish larvae from high-throughput axial-view in vivo imaging. *Biomed. Opt. Express* **8,** 2611–2634. https://opg.optica.org/boe/abstract.cfm?URI=boe-8-5-2611 (May 2017).
3. Guo, Y., Xiong, Z. & Verbeek, F. J. An efficient and robust hybrid method for segmentation of zebrafish objects from bright-field microscope images. *Machine Vision and Applications* **29,** 1211–1225. ISSN: 1432-1769. https://doi.org/10.1007/s00138-018-0934-y (Nov. 2018).
4. Hamelink, J. Camera Parameter Optimization for the VAST using CMA-ES (2022).
5. Howe, K. *et al.* The zebrafish reference genome sequence and its relationship to the human genome. en. *Nature* **496,** 498–503 (Apr. 2013).
6. Kimmel, C. B., Ballard, W. W., Kimmel, S. R., Ullmann, B. & Schilling, T. F. Stages of embryonic development of the zebrafish. en. *Dev. Dyn.* **203,** 253–310 (July 1995).
7. Microsystems, L. *Leica DM6000 B Fully Automated Upright Microscope System for Cutting-edge Life Science Research* https://www.leica-microsystems.com/products/light-microscopes/p/leica-dm6000-b/.
8. Paço, M. *Combining Patterned Optogenetics Using a Digital Micromirror Device and Light-sheet Whole-Brain Imaging in Behaving Fish* PhD thesis (Nov. 2017).
9. Pinkard, H. *et al.* Pycro-Manager: open-source software for customized and reproducible microscope control. *Nature Methods* **18,** 226–228. ISSN: 1548-7105. https://doi.org/10.1038/s41592-021-01087-6 (Mar. 2021).
10. Postlethwait, J. H. *et al.* Vertebrate genome evolution and the zebrafish gene map. en. *Nat. Genet.* **18,** 345–349 (Apr. 1998).
11. University, L. *Zebrafish models for disease and environmental stress* https://www.universiteitleiden.nl/en/news/2015/01/zebrafish-as-model-organism. 2016.
12. Zwart, R. *ZVizApp* https://zebrafish.liacs.nl/#/gridTab.

7 Appendix

Link to Github repository containing all of the software (read only).
Link to LIACS' GitLab repository containing all of the software.
Higher resolution version of fig 1.