Florian Hubner

Development of an interface for fast read out of high-resolution two-photon images

Master's Thesis

Graz University of Technology

Institute for Theoretical Computer Science
Head: O. Univ.-Prof. Dr. Wolfgang Maass

Supervisor: O. Univ.-Prof. Dr. Wolfgang Maass

Graz, September 2013

This document is set in Palatino, compiled with pdfLaTeX2e and Biber.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Date Signature

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Datum Unterschrift

[1] Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Abstract

Novel two-photon microscopes are able to produce images with high resolution and frame rate. Usually the images produced by this microscope are saved to disk and then analysed offline. In the course of this thesis an interface was developed which allows to send these images over the network. This way analyses can be done at the same time as the images are acquired. The online analysis of such images allows to develop experiments where immediate feedback of the activity of certain neurons is given back to the monitored animal. For the purpose of the network interface a packet structure was defined which specifies how the images are sent over the network. Using this encoding a sender was implemented on top of ScanImage[12]. Also a receiver was implemented in MATLAB which is able to reassemble the images from the data received over the network. To show that the interface works for real experiments some tests were made and application which use the interface were developed. The last application, the auto focus, which was in the beginning also meant to use the network interface turned out to be more useful if directly implemented as a plugin for ScanImage.

Keywords: ScanImage, two-photon imaging, real time, calcium traces

Zussammenfassung

Neuartige two-photon Mikroskope sind in der Lage Bilder mit hoher Auflösung und Bildrate zu machen. Normalerweise werden diese Bilder auf der Festplatte gespeichert und dann offline analysiert. Im Zuge dieser Arbeit wurde eine Schnittstelle entwickelt, die es möglich macht die Bilder über ein Netzwerk zu senden. Dadurch können die Daten analysiert werden während noch Bilder mit dem Mikroskope gemacht werden. Diese online Analyse erlaubt es Experimente zu gestalten in denen Rückmeldung über die Aktivität bestimmter Neuronen an das Tier zurückgegeben wird. Für die Netzwerkschnittstelle wurde eine spezielle Packetstruktur spezifiziert die genau bestimmt wie die Bilder über das Netwerk geschickt werden. Mit Hilfe dieser Kodierung wurde dann ein Sender für ScanImage implementiert. Außerdem wurde ein Empfänger entwickelt, der die Daten die über das Netzwerk empfangen werden wieder zusammensetzt. Um zu zeigen, dass die Schnittstell auch in echten Experimenten funktioniert, wurden Tests gemacht und Anwendungen entwickelt, die die Schnittstelle verwenden. Die letzte Anwendung, der Auto Focus, sollte am Anfang auch die Netzwerkschnittstelle verwenden. Es stellte sich jedoch heraus, dass er besser zu bedienen ist wenn er direkt als Plugin für ScanImage implementiert wird.

Schlüsselwörter: ScanImage, two-photon imaging, real time, calcium traces

Contents

List of Figures

1. Introduction

The brain is a very interesting topic of research for many people. It can perform very complex task, it is flexible and can adapt to the environment, and it is at the same time very energy efficient. To better understand the brain scientist developed different methods to analyse what is going on in this complex organ. These methods reach from observing and measuring single cells to behavioural studies or psychoanalyses, they can be inversive or non-inversive, in vivo or in vitro, on a low level or on a high level, and so forth. All these different approaches are necessary to understand all the different working levels of the brain.

From the view point of computer science it is also very interesting to find out how the brain works because it allows to develop new kinds of artificial intelligence (AI) and technologies. This is important because there are many tasks which are very easy for humans but very hard for computers. For example teaching a robot to navigate through a building is very complicated. Another branch of computer science where brain inspired AIs are interesting is computer vision. Classifying and categorizing objects in images is a topic of many publications[1, 3, 4] and understanding the brain can help to come up with new approaches.
There are also already researches which try to develop hardware based on the function of the brain[7, 16]. This can lead to a new, very different, kind of computers which are able to perform tasks that todays computers can not perform.
There are many more areas where brain inspired hard- and software are useful, but to be able to develop them we need to better understand how the brain works.

For a better insight into the brain novel techniques were developed during the last few decades which allow to directly measure the activity of neurons.

Some examples are functional magnetic resonance imaging (fMRI), measurement with electrodes in the brain, and two-photon imaging[2, 14, 15]. A quick introduction to two-photon imaging can be found in Section 1.1. With these methods the behaviour of groups of neurons can be observed more precisely and correlations to behavioural aspects can be found. This makes it possible to examine how the brain codes different types of information, from low level information, like sensory input or motor output, to higher level information, like memory.

The focus of this thesis lies on the work with images made with a two-photon microscope. The development has been done in the lab of Daniel Huber. The goal is to be able to send images acquired with the two-photon microscope in real time over the network. To achieve this I developed a network interface which is described in Chapter 2. Before implementing the interface I specified a packet structure which exactly defines how the images are sent over the network (see Section 2.2.2). With this encoding defined it is possible to implement a sender and a receiver. The details of the implementation of those can be found in Section 2.3. The sender has been developed as a plugin for ScanImage[12]. The receiver has been implemented in MATLAB but because of the defined packet structure it would be possible to use any programming language to implement the receiver. It offers a nice way for users to add their own analysing functions. Some test are presented which show that the implementation of the interface is fast enough to be used in an experimental setup. The results of the tests can be found in Section 2.4.

Additionally to the tests I developed some application in collaboration with Mario Prsa from the lab of Daniel Huber. An exact description of the applications I developed can be found in Chapter 3. The first two of these application use the network interface. They should in combination with the test from Section 2.4 show that the interface I developed for this thesis can be used in real experiments. The first application is called the real-time tracer. It allows to extract fluorescence traces from the images sent over the network. In Section 3.1 it is described how this tool works. On top of the real-time tracer an has been implemented by Mario and me with which it is possible to move a robot arm depending on the activity of neurons of a mouse (see Section 3.2). The third one, the auto focus, has also been planned to use the interface but it turned out that is makes more sense

to directly implement it as a plugin for ScanImage. Its purpose is to help the experimenter to get to the same position in the brain if experiments on different days are done. A discussion of the problems and how their are solved can be found in Section 3.3.

In the last chapter a conclusion for the thesis can be found. It summarizes findings of these thesis. Additionally some ideas for improvements and future work are mentioned. In the appendix a user manual can be found. It gives step by step instructions how the interface is installed, explains how it is used and shows some programming examples.

1.1. Two-photon calcium imaging

All the data for this thesis is based on images acquired with a two-photon microscope in brains of mice. For a better understanding of this thesis it is useful to know how these images are obtained. It will make clear what the problems are, and why some decisions are made.

1.1.1. Advantages

Two-photon imaging allows to observe the activity of a group of neurons simultaneously. By monitoring several neurons of a specific brain area conclusions can be drawn about what the animal is doing. It also allows to look at neurons of one specific type which helps to understand what different cell types in the brain are responsible for. Another very important advantage of this technique is that images can be made in alive (in vivo) and awake animals. Imaging in awake animals is a very important feature as cells of dead or anesthetized can behave differently than the cells of awake animals. analysing the brain also becomes more flexible in vivo because some errors can be found and correct more easily as in vitro. Newer microscopes are able to make images with a high resolution and high frame rates at the same time. This generates a huge amount of data that can be used for studying the brain.

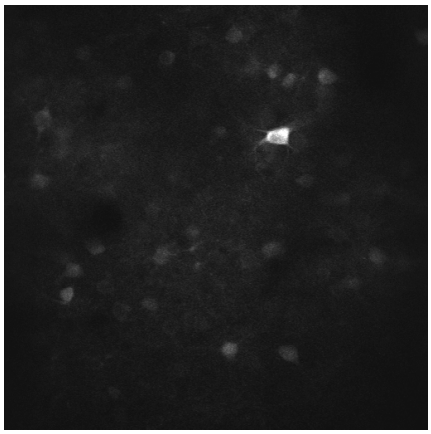

Figure 1.1.: Example of an image taken with a two-photon microscope.

1.1.2. The imaging process

The whole process of acquiring images with this technique is based on a laser which excites special fluorescent molecules to emit photons. These photons are then measured and converted into a single pixel value. The laser is then moved a bit and another measurement is taken. This is done until a whole image is ready after which the laser gets back to the starting position and starts a new image. In Figure 1.1 an example for such an image is shown where several neurons with different levels of activity can be seen.

To be able to measure the activity the probability that a molecule emits photon depends on the calcium concentration inside of it. Calcium is a very good indicator of how big the voltage between inside and outside of a neuron is which represents the activity of the cell. It is also important that the fluorescent molecules are only inside the neurons which you want to see on the image. To achieve this there are mainly two methods. The first is to inject a special dye[14] which the neurons will absorb. The second method is to modify the gens[9] of the animal so that it expresses certain kinds of proteins that are able to fluoresce.

Important factors for the quality of the image are temporal and spatial resolutions. The necessary temporal resolution is achieved with the laser

which is capable of firing very short burst of photon very fast. Therefor pixel values can be gather very fast which leads to a high frame rate for the image. Spatial resolution is achieved by the way how the molecules are excited. It always takes two photon of the laser for the molecule to emit one photon. That is why it is called two-photon imaging. When using two photon for the excitation it is possible to focus the laser on a specific part of the tissue. If only one photon would be used all neurons on the path of the laser would emit photons and it would not be possible to measure one specific point in the tissue.

2. Interface for fast readout

The primary goal for this thesis is to find a way to send the images produced by the two-photon microscope over the network. The microscope is controlled by the acquisition computer where a special acquisition software is running. It will send the images to another computer that will then be able to work with the data. Using a network based interface has the following advantages:

- Move analysis to more powerful machines
- Decouple acquisition from analysis
- Independence of programming language

The data produced by a two-photon microscope can be very large so if it is possible to send the data to a computer with high computational power the data can be analysed faster and more complex algorithms can be used. Processes on the acquisition machine , like analysing the images, can slow down the acquisition of images which in some cases can lead to errors in the images or in the worst case even to a crash of the acquisition software. When using dedicated machines for the analysis this problem can be overcome. The interface defines a clear structure of how the data have to be encoded. Therefor it is possible to use an arbitrary programming language for the implementation of the readout.

2.1. Setup

Figure 2.1 shows a schematic of the general setup for the imaging. Note that the implementation for this thesis are manly developed in the lab of Daniel Huber at the University of Geneva but the schematic should also apply to setups in different labs just the animal used for the experiments

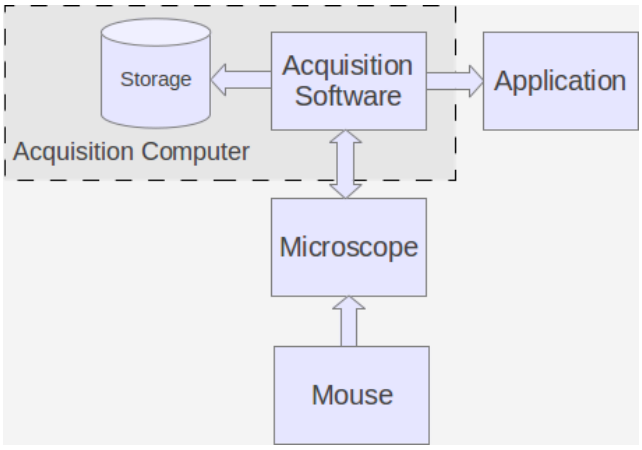

Figure 2.1.: Schematic that shows the different parts of the imaging setup.

may be different. The first part involved in the setup is the mouse (or animal used for imaging). Usually its head is fixed that the microscope can take proper images. The microscope contains the hardware which is necessary to take two-photon images. The hardware consists of the laser, several optical devices to control the beam, devices that measure the photons emitted from the fluorescent molecules, and so forth. Additional equipment can be added which can be necessary for the experiment. To control the microscope an acquisition computer is needed. It contains hardware that is able to control the different devices of the microscope so it is able to change properties of the imaging process. A special acquisition software is also needed to allow the user to control the whole imaging process. The software will store the images usually on the hard disk, but the acquisition software should be able to not only store the data to disk but also send it over the network. So a network interface has been developed which allows exactly this. Another computer can then use this interface to read out the data an work with them.

2.1.1. ScanImage

The acquisition software to control the microscope for which the interface was developed is called ScanImage[12]. It is implemented in MATLAB in an object oriented style. ScanImage uses special hardware to communicate with the microscope and also offers a graphical user interface (GUI) which allows to control every aspect of the imaging process. The GUI actually consists of several small windows where each is responsible for a specific task. For example one window allows to control where and when the images are stored, another one controls the properties of the image, and so on. A very important feature of ScanImage is that it allows developers to add user defined function and plugins. Using plugins it is very easy to connect the interface for the fast readout to ScanImage. A more detailed explanation of the GUI, user defined functions, plugins, and generally about ScanImage can be found on the project homepage[6].

ScanImage offers three different modes how images can be made.

- **FOCUS**
  This mode is to take a first look at the neurons.
- **GRAB**
  With this mode a set of images can be acquired.
- **LOOP**
  The LOOP mode allows to do several GRABs.

The first mode allows to go to the region where the imaging should take place. When imaging an animal the first time the FOCUS-mode can be used to check if the neurons can be seen properly and to find a suitable region for the imaging. After finding a suitable region the GRAB-mode can be used to get first images which will be saved in a single TIF file to the disk. It can be configured how many images to take, if they should be averaged, how many slices (images in z-levels) to take, and where the images should be saved. For experiments it is useful to utilize the LOOP-mode. It allows to do several consecutive GRABs. In the easiest case the GRABs are just made one after another. For each GRAB one file is saved in a specified folder. If making long acquisition this is necessary otherwise the images can get to big to handle them. The LOOP-mode can also be configured in a way that

when a external trigger occurs the next GRAB starts. This can be used in an experiment to start a new acquisition for every trial.

2.2. Interface

For the reasons already mentioned above a defined interface is needed. It should make clear how the data are encoded and sent over the network. To be able to do this one has to exactly understand how the data look like. It is not enough to just send the image over the network. Additional information is required which is needed to work with them. This information is called meta data. It contains things like width and height of the image, a timestamp of when the image was acquired, the number of the image, and several other parameters of the imaging process. Fortunately ScanImage already provides all of the necessary information.

The next important thing to think of is how the data are sent over the network. Computer networks can be very heterogeneous, assume many different forms, and can strongly vary in size. To control the traffic in this networks standards have been developed which exactly define how data have to be encoded before sending them. This way all devices in the network know how to understand the data they receive. These standards are called protocols. The most famous are probably TCP and IP but there are many more. To define the interface it is necessary to decide what protocol should be used.

2.2.1. TCP vs UDP

For the transportation of the kind of data used here there are mainly two options that can be used, the Transmission Control Protocol(TCP) and the User Datagram Protocol(UDP). They are built on top of the Internet Protocol(IP) and define how two computers exchange data. Both have different features which can be seen in Table 2.1.

	TCP	UDP
Name	Transmission Control Protocol	User Datagram Protocol
Connection	connection oriented	connectionless
Ordering	correct	arbitrarily
Packet loss	packet is resent	ignored

Table 2.1.: Properties of TCP and UDP

TCP The Transmission Control Protocol if probably the most used protocol which handles the data exchange between computer. It is a connection oriented protocol, that means before data can be sent from one client to the other client it builds up a connection between them and when a session is finished the two clients disconnect. If data are sent over the network it can happen that some packets (data pieces) can be faster then others so the right ordering of the packets is not guaranteed, TCP handles this problem and reorders them. Another problem that can occur is that some packets can be lost on the way from one client to the other. When this happens TCP sends a request to resend this packet.

UDP The User Datagram Protocol is much simpler than the other protocol. It does not build up a connection between computers but rather just sends the packets out into the network. It also does not care if a packet arrives at the other client or if the packets are received in the correct order. At first this sounds as if UDP is useless compared to TCP but the advantage is that it is much faster because it has much less overhead. While TCP needs an extra procedure to build up a connection, which takes some time, UDP can start sending data right away. Also the resending of packets is very time consuming because for every packet a client receives it has to send back a message to the other client that it received it. The reordering of the packets has to be taken into account by the implementation of the interface.

To sum everything up it can be said that TCP should be used when it is

important that all pieces of the data are received correctly. UDP on the other hand should be used if speed is important and the loss of some data bits is not that crucial. In the case of this thesis UDP is chosen because the speed is much more important than the correct receiving of the data. As the frame rate of the images is usually pretty high there will not be to much changes from one image to another and missing parts could be interpolate from the previous and the next image.

2.2.2. Packet structure

Now that the shape of the data is known and it is decided which protocol is used for transmission it can be defined how the data are encoded. First they have to be split up into smaller pieces because the packets sent over the network may only have a limited size. Additionally a header will be added (see Figure 2.2) to each piece to be able to reconstruct the original data, and to know what data are received. The first four bytes of the header determine the type, dependent on this the header can contain additional information. There are four possible packet types.

- **FRAM** Indicates that the data received are image data. In this case the header also contains the number of the frame and a number that says which part of the image this data piece is.
- **META** This type of packet contains the meta data of the imaging process. It has no additional information in the header.
- **DONE** It indicates that a frame is completely sent. The data section of this packet is empty.
- **QUIT** Tells the client that the acquisition software has been closed. The data section is also empty.

These packets can now be sent over the network using UDP. The receiving client will then know what data it is, how it should handle them, and how it can reconstruct the images.

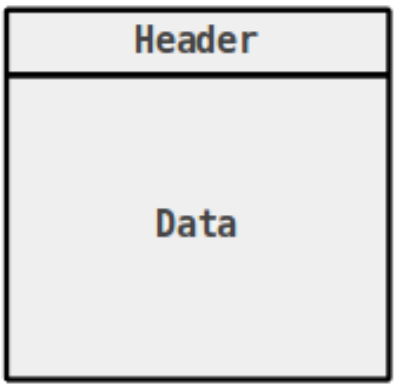

Figure 2.2.: Encoding of the data

Figure 2.3.: Interface with all its components

2.3. Implementation

The next step after deciding how the data are sent over the network is to implement the interface. Because ScanImage is coded in MATLAB it is also chosen as the language for the implementation of the interface. Another reason why MATLAB is used is that it already offers many functions for image processing which will be useful for the analysis of the data.

The interface consists of two parts, the sender who encodes the data and the receiver who decodes and reassembles the data. This is necessary because a network based interface is used. The sender has to communicate with ScanImage to get the images and the meta data. The receiver should provide an easy way to read out the reassembled images. Figure 2.3

2.3.1. Sender

As mentioned before the sender gets the images and the meta data from ScanImage. It is implemented as a plugin for ScanImage. This means it is actually just a function which is called when certain events happen in ScanImage. The sender then, depending on the event that occurred, performs the necessary tasks. The events that are important for the sender are:

- *appOpen*
  Indicates that ScanImage has started.
- *appClose*
  Occurs when ScanImage is closed.
- *frameAcquired*
  When an image (frame) is ready this event indicates it.
- *acquisitionStarted*
  This event indicates that the acquisition has started.
- *acquisitionDone*
  All images are done when this event occurs.

At the beginning, when ScanImage is started, the sender needs to initialized the network components that will be used during the communication. For this an IP address has to be provided. How this can be done will be shown in Section A. After that the plugin waits until ScanImage starts an acquisition which is indicated by the *acquisitionStarted*-event. In ScanImage a start of an acquisition is always when a new GRAB starts that means either a GRAB is directly started by the user or in LOOP-mode when the next GRAB starts. The FOCUS-mode will not trigger this event. The sender then reads the meta data and sends them to the receiver using a META-packet. Then one after the other frame will be ready which is indicated by *frameAcquired*. Each frame gets partitioned into several data pieces, put into a packet of the type DATA, and is then sent over the network. When all the images are done the event *acquisitionDone* is triggered which makes the plugin to send a DONE-packet. Finally, if ScanImage is closed, the used network components are closed and everything gets cleaned up.

2.3.2. Receiver

The second part of the interface is called the receiver. It reads incoming packets from the network, unpacks them and reassembles the data. Depending on the type of the packet it performs different tasks but before it can do this some initializations have to be made. So at the beginning, when the receiver starts, it sets up the necessary network components. After that it waits for packets from the sender. The first packet which will be sent is of the type META. The receiver needs this because it contains important information about size and number of images which are going to be received. The next packets that are sent are FRAM-packets. Each of them contains a segment of the image. To be able to reassemble the image additional information is provided in the packet header. It tells the receiver to which frame the data in the packet belong and which part of the image it is. When a frame is ready the receiver triggers an event called *frameComplete*. It allows the user to define a function which will be called every time this event happens. This way the user is able to analyse the data that are received. For a detailed explanation of how to define these functions see Section A. The DONE packet indicates that a frame is completely sent. The receiver uses this type of packet to find out if some packets are lost. When all frames have been received the receiver waits for another packet. The two types of the packets it expects are either META or QUIT. If a META packet is received the process described before is performed again. A packet of the type QUIT indicates that ScanImage was closed and triggers the receiver to also shut down. Similar to the sender it will perform some cleaning up before it will completely terminated.

Note that the *frameComplete* event does not necessarily mean that all the data were received because it can happen that some packets are lost. In this case the respective part of the image will be wrong. At the current version this case is just ignored, because only a few packets are lost and usually the images are averaged during the analysis which will correct for the lost image parts. The decision for ignoring this case is made because for the first implementation speed is most important and sophisticated methods for recovering the lost parts would need time.

2.4. Benchmark

For the last section of this chapter some speed test were made to evaluate how much time is needed to reassemble the images and how much time is left for analysis. The test were made using a custom made acquisition computer with which is also used for the experiments. It sends the data over a small local network to an Acer Aspire 5935g laptop with an Intel Core 2 Duo CPU T9550 with 2.66 GHz and 4 GB of RAM. Compared to the computers usually used for the analyses of the data the laptop is rather weak so the data represented here that depend on the power of the computer are a rather pessimistic estimate. The interface is designed to be used in local networks and not for the use over the internet, therefor the test are done using the network in which the interface will be mainly used. Three different properties are measured in the benchmark:

- **assembly time**
  Time needed to reassemble the images received from the sender.
- **packet loss**
  Number of packets that are lost in the network.
- **packet loss vs processing time**
  Percent of lost packets over the time used for analysing the data.

The data are measured using the highest possible frame rate of $28.84Hz$ for 512×512 images. 10000 images are sent which takes 341 seconds for the chosen frame rate. Figures 2.4 - 2.6 show the outcome of the benchmarking.

In the first figure it can be seen how much time it takes to reassemble each of the 10000 frames. The mean assembly time is roughly at $10ms$ and there are peaks up to about $22ms$. These peaks can appear because of the different delays each packet has. Another reason why they can show up is that a process on the receiving computer may need some CPU time thus increasing the time needed to reassemble the image. Note that the assembly time depends on the power of the computer.

The data shown in Figure 2.5 represent how many packets are lost in the network. For this figure no analysis of the data are done meaning that the

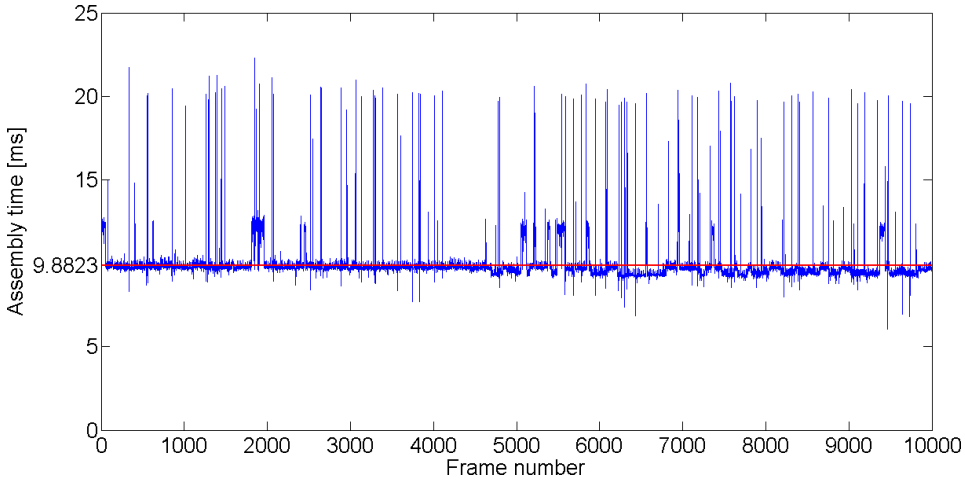

Figure 2.4.: Time needed to reassemble the images. The red line shows the mean assembly time of 9.8823*ms*.

images are only sent over the network, reassembled and then discarded. It clearly shows that the packet loss for small networks is negligible.

The next question is if the packet loss changes if some analysis are done. This is tested and shown in Figure 2.6. Roughly the first 10*ms* are needed to reassemble the image which is indicated with the gray area. The time that should be left for the analysis can be computed using the frame rate, and the assembly time. When this is done for these data the time left should be about 24.8*ms* which perfectly fits the data shown. If the analysis takes more time the number of lost packets will increase strongly. This is because the packets received are first stored into a buffer. The receiver takes the packets from this buffer and reassembles the images. When an image is ready the analysing function is called. If this function is not fast enough to handle the frame rate at which the data are sent the buffer will overflow and therefor some packets are discarded.

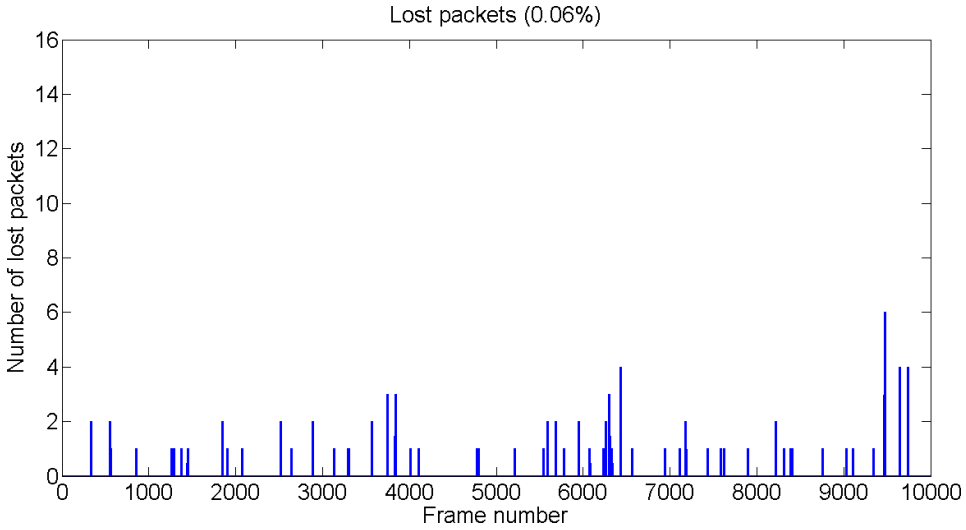

Figure 2.5.: Packet lost in the network when no analysis are done. The percentage of lost packets is 0.06%

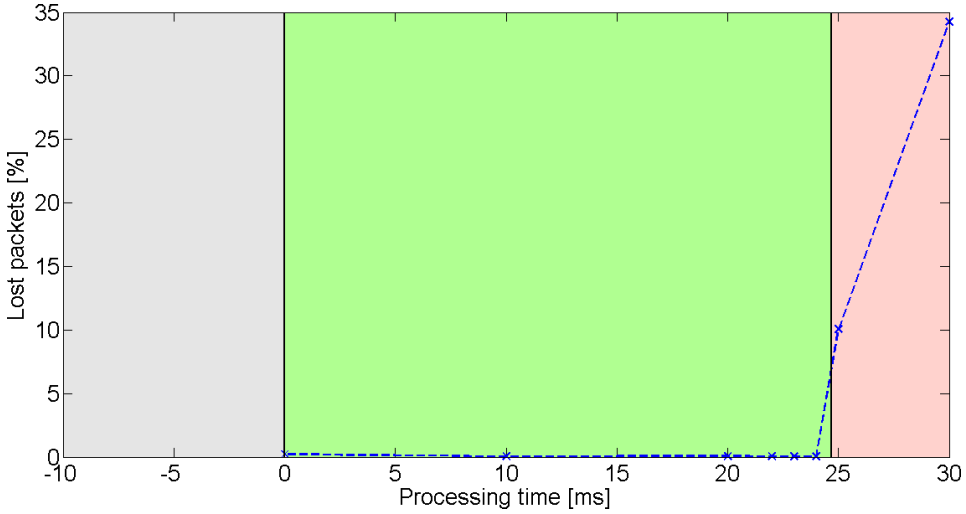

Figure 2.6.: Percentage of packets that are lost for different analysing times. The gray area shows the time needed for the assembly of the image, the green area is region is the acceptable time for image analysis, and the red area shows the time where data are lost because the analysis can not handle the frame rate at which the images are produced. Blue crosses are measurement points.

3. Applications

In the previous chapter it was already shown that the interface should work but this was under some benchmark conditions which may not resemble real world conditions. To see if it also works for experiments additional applications have been developed. They have been tested under real experimental conditions and with real mice. It also shows different ways in which the interface can be used.

The application have been developed in close collaboration with Mario Prsa who works at the lab of Daniel Huber. They have been also mainly developed for this lab. So the applications might not work from scratch in other labs and may need some changes in the code.

3.1. Real-time tracer

The first thing to do for the analyses is to extract the activity of some neurons from the images. This application is able to do this in real time as it receives the images using the network interface. It extracts the activity of some predefined neurons and generates a signal which can then be used for further analyses. To make it easier to use it also offers a GUI.

3.1.1. GUI

The GUI shows all the information which is necessary for the user and allows to adjust some parameters for how the signal is computed. In Figure 3.1 a screenshot of the GUI can be seen. It can work in two different modes. The first one is to define the regions of interest (ROI) which are the neurons

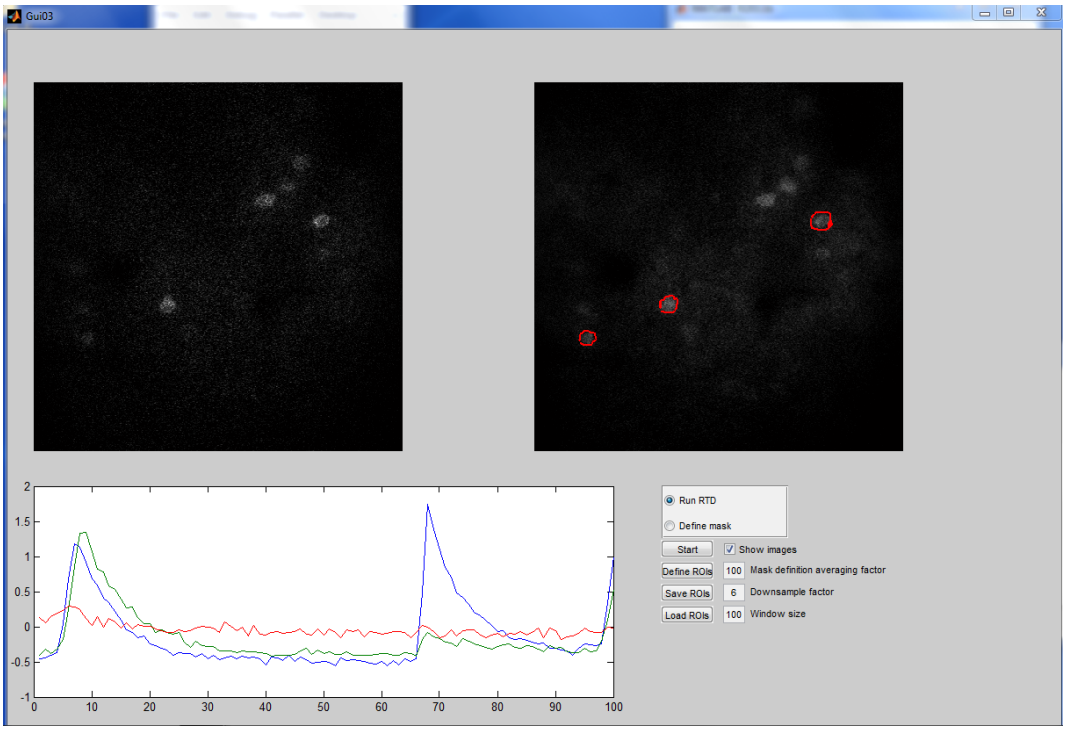

Figure 3.1.: Screenshot of the GUI of the real-time tracer

from which the activity should be extracted. In this mode the upper left plot shows an averaged image of all the images it receives. It can then be used to define which are the interesting neurons. The second mode is used to extract the activity of the neurons from the images. Here the upper left plot shows the images that are received over the network. The upper right plot shows a moving average of the received images and the ROIs that have been defined. In the lower plot the extracted activities of the different neurons can be seen. The lower right area of the GUI is where the parameters and modes can be configured.

3.1.2. Computation of the activity signal

For further analysis a signal has to be extracted from the imaging data. Unfortunately the molecules that allow to measure the calcium concentration with the microscope may vary over different areas and also the setting of the imaging parameters may change the intensity of a neuron. But the signal that is produced from the images should represent the activity of the neuron which is independent from these factors. Thus the signal has to be normalized in a way that the activity from different areas and from different images can be compared. This is done in three steps.

- Averaging
- Masking
- Com

The first step is necessary to get rid of any kind of noise which may be produced by the microscope or to compensate the data which might get lost over the network. After that the average image in combination with masks which indicate the ROIs can be used to extract a raw signal. This is done by just computing the mean pixel value inside each ROI. The signals produced this way still depend on several factors like the setting for the power of the laser and the concentration of calcium indicating molecules. To make the signal comparable a sliding window is used to determine the base activity which is then subtracted from the raw value. The result is then divided by the base value to normalize it. This way a signal is produced which can be used for further analyses.

3.2. Robot arm control

This application demonstrates the capabilities of the interface and also of the real time tracer. It shows that it is possible to extract the activity of some neurons and transform into movements of a machine. Controlling devices using neuronal activity will be important for further experiments where the mouse is trained to control some device using its mind.

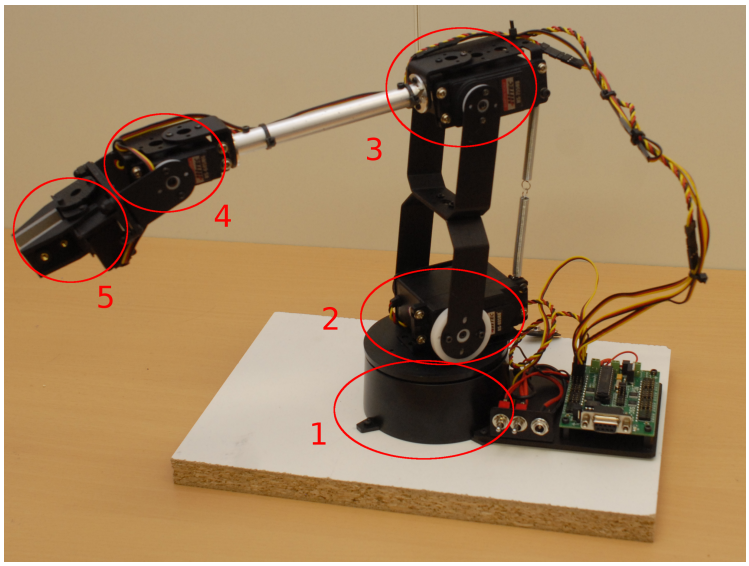

Figure 3.2.: Robot arm with all controllable joints marked.

Figure 3.2 shows the robot arm that is used for this application. It has a grabber and five controllable joints which are marked in the figure. The first one allows to rotate the arm. The next one allows to move the arm back and forth. Joint 3,4 are used to move the arm and the grabber up and down and the last joint opens and closes the grabber. For the application only the joints 1,3,4 are used because it is enough illustrate the purpose of this application. The three joints are controlled by three neurons that are selected by hand. To extract the activity of the neurons the real-time tracer is used. The produced signal can not be directly used as input for robot arm so it has to be transformed a bit. The transformation consists of two parts. First it is filtered using a Gaussian filter to make the movements smoother. Without the filter the changes of the signal are very fast and the movements would be very abrupt. The filtering also introduces a small delay but this is in about the same range as the delay from the movement onset in the brain to the real movement in mice. After that the signal is transformed with a hyperbolic tangent so that the values are between -1 and 1. The resulting signal can then be used to control the arm.

On the accompanied CD a video can be found which shows the application

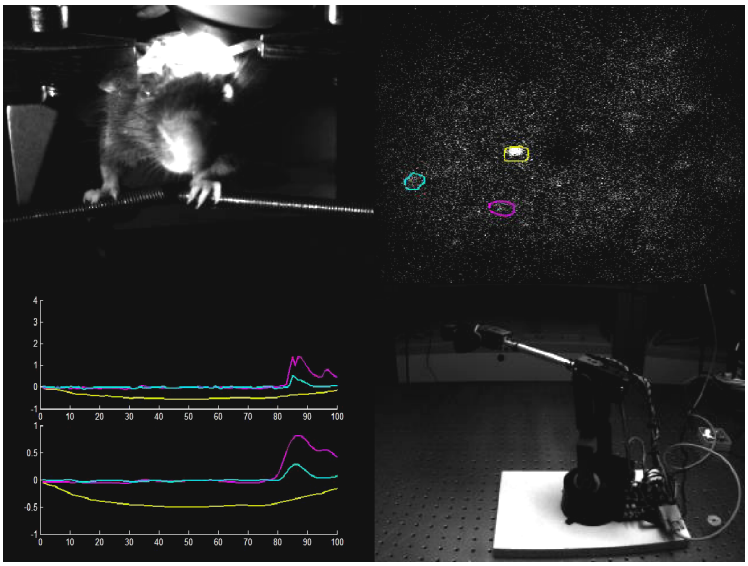

Figure 3.3.: Example frame of the robot arm video. Upper left part shows the actions of the mouse. The upper right part shows images made with the two-photon microscope. The lower left part shows the extracted calcium trace (upper plot) and the transformed signal (lower plot). The lower right part shows the movement of the robot arm.

in action. In Figure 3.3 a frame from the video can be seen. It consists of four parts. The upper left part shows what the mouse is doing at the moment. The upper right one shows the activity of the neuron. It is also indicated which neuron corresponds to which signal in the traces. The lower left plot shows the extracted activity from the real-time tracer and the transformed signal that is used to control the robot arm. The last part shows the movement of the arm. Note that in the video the delay is removed for a better comparability of the four sub videos.

3.3. Auto focus

The auto focus is a tool that should simplify the process of the experiments. Often it is the case that images are recorded on different days but in the

same area. Therefor at the beginning of each experiment the same area has to be found by eye. Of course this is not very exact. That is where the auto focus should help. First the microscope has to be roughly set to the area where the images on the last days have been taken. After that it scans the current location and tells the experimenter where the microscope has to be moved. It is also important to make it easy to use and not to make it to slow. In the beginning it was planned that it should also use the network interface but it turned out that the problem is more complex than thought and it can be made more user friendly when ran on the same machine. It is still in a very early state of development and there are some issues which have to be taken care of.

3.3.1. Goal

To be able to realize the auto focus it first has to be clarified what exactly should be computed and what is needed to do so. As mentioned above the auto focus should determine the offset of the current location to a reference location. The locations are given in form of images. So the task is to find out how the image of the current location has to be shifted to match the reference image best. In other words, the shift which maximizes the match of the two images has to be found. When a single image is used as reference it is possible to determine the offset in x- and y-direction but the auto focus should also be able to find the z-offset. Therefor a single reference image is not enough but a whole stack of images is needed. The reference stack is actually just a discretized form of a 3D volume. To determine the offset in all direction again the shift that produces the best match has to be found.

Now the offset in images coordinates, that means pixel, is known. The next thing to do is to transform these coordinates into the coordinate system of the microscope. Unfortunately the transformation from the image coordinate system to the coordinate system of the microscope is not just a simple scaling. There are several other transformation that has to be done to get from one coordinate system to the other which are a rotation so that the axis are aligned, different scalings along x- and y-axis, and a translation so that the origins are aligned. All these transformations can be done using a single matrix multiplication. So the problem of how to transform between the

two coordinate systems has reduced to finding out a transformation matrix. To do this the auto focus has to be calibrated at the beginning of each experiment. Every time something in the setup of the microscope changes, which also includes mounting an animal into the microscope, the auto focus has to be recalibrated because the transformation between the coordinate systems might change.

In conclusion there are two tasks that have to be solved. The first task is to be able to compute offsets between images. For the second task a calibration has to be done. It allows to compute the transformation matrix which can be used to transform from the image coordinate system to the microscope coordinate system.

3.3.2. Offset computation

The goal of the first task is to find a way how the offsets in all directions can be computed. For the beginning it is easier to ignore the z-coordinate and just concentrate on finding out how the offset between two images can be computed. So the simplest way of doing this is to take one image, shift it over the other image and compute a measure of similarity between the images. Exactly this is what the normalized cross-correlation[8] does. It is a measure of similarity between two images depending on an offset which is applied two one of the images. When the normalized cross-correlation is computed for the images a matrix (Figure 3.4) will be produced which contains values between 1, if the images are the same for the given offset, and -1, if the images are the exact opposite. Therefor the indices of the maximum value of that matrix can be used to compute the offset.

The next step is to extend this approach in a way that also the offset in z-direction can be determined. The simplest way to do this is to compute the normalized cross-correlation for each image of the stack. After that the z-offset in image coordinates can be determined by finding the index of the image that produces the highest value for the normalized cross-correlation.

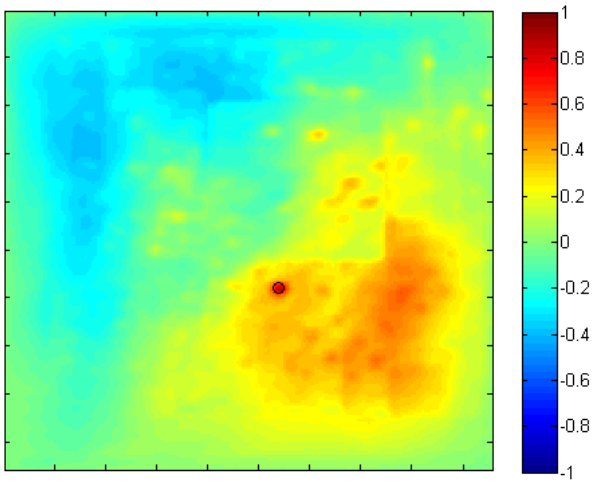

Figure 3.4.: Example of the normalized cross-correlation of two images. Dark blue areas are offsets for which the normalized cross-correlation is low, while red areas indicate offsets with high correlation. The black circle shows the maximum correlation.

3.3.3. Transformation matrix

The second important task which has to be solved for the auto focus is the transformation from image coordinates to microscope coordinates. The images coordinates consist of the pixel indices in x- and y-direction and the image index from the stack in z-direction. The coordinates of the microscope are in μm and can be controlled using ScanImage. Generally these two coordinate system are not aligned.

To simplify the task of transforming to the microscope coordinates the z-direction is handled separately. It is possible that there might is some rotational transformation between the z-axis of the two coordinate systems but this rotation is rather small and can not be recognized by eye. Additionally it is not possible to change the angle of the z-axis of the microscope without changing the setup up. Therefor the z-direction is computed by multiplying the z-offset in image coordinates, which is just an index offset, by the step size in μm between the stack images. The step size is automatically saved

by ScanImage inside the stack file and can be easily accessed.

The remaining problem is to find a transformation from one two-dimensional coordinate system to another. To be able to express this transformation in form of a simple matrix multiplication so called homogeneous coordinates[5, p. 2] are introduced. Homogeneous coordinates extend, in this case, the two dimensional coordinates by a third one. So the transformation from the image coordinate system to the microscope coordinates can be written as

$$\begin{pmatrix} x_m \\ y_m \\ w_m \end{pmatrix} = \mathbf{T} \begin{pmatrix} x_i \\ y_i \\ w_i \end{pmatrix} \tag{3.1}$$

where x_m, y_m and w_m are the homogeneous microscope coordinates, x_i, y_i and w_i are the homogeneous image coordinates and $\mathbf{T}$ is the transformation matrix. $\mathbf{T}$ is a three 3×3 matrix and can be written the following way

$$\mathbf{T} = \begin{pmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{pmatrix}$$

If all the entries can be arbitrarily nine equations would be needed to determine all of them. Additionally the transformation with this matrix can produce results that may not be fit the problem. Actually only the following basic transformations can occur

- translation
- scaling along the x-axis
- scaling along the y-axis
- rotation

These transformations are all simple affine transformations. Therefor the transformation matrix can be reduced to

$$\mathbf{T} = \begin{pmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ 0 & 0 & 1 \end{pmatrix}$$

Now that the shape of $\mathbf{T}$ is known the next step is to find out the entries of the matrix.

To determine the values of $\mathbf{T}$ a calibration has to be done. This means that some images where the offset in microscope coordinates is known. From these images the offset in image coordinates can be computed which gives pairs of corresponding offsets. These correspondences can be used to compute the values of $\mathbf{T}$. The formula how to compute the entries of the matrix can be found by using the discrete linear transformation(DLT)[5, p. 88ff][1]. As a starting point equation 3.1 is used, rewritten in vector notation, and transformed.

$$\begin{aligned} \mathbf{x}_m &= \mathbf{T}\mathbf{x}_i \\ \mathbf{x}_m - \mathbf{T}\mathbf{x}_i &= \mathbf{0} \end{aligned}$$

For further rewriting a new notation for the transformation matrix is introduced.

$$\mathbf{T} = \begin{pmatrix} \mathbf{t}^{1\mathsf{T}} \\ \mathbf{t}^{2\mathsf{T}} \\ \mathbf{t}^{3\mathsf{T}} \end{pmatrix} = \begin{pmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ 0 & 0 & 1 \end{pmatrix}$$

[1] The approach here is slightly different from the one in the reference. But the goal of DLT is to get a formula that is linear in the parameter that should be found which is also achieved here.

Where $\mathbf{t}^{i\mathsf{T}}$ is the i-th row of $\mathbf{T}$.

$$\mathbf{x}_m - \begin{pmatrix} \mathbf{t}^{1\mathsf{T}} \\ \mathbf{t}^{2\mathsf{T}} \\ \mathbf{t}^{3\mathsf{T}} \end{pmatrix} \mathbf{x}_i = \mathbf{0}$$

$$\mathbf{x}_m - \begin{pmatrix} \mathbf{t}^{1\mathsf{T}}\mathbf{x}_i \\ \mathbf{t}^{2\mathsf{T}}\mathbf{x}_i \\ \mathbf{t}^{3\mathsf{T}}\mathbf{x}_i \end{pmatrix} = \mathbf{0}$$

$$\begin{pmatrix} x_m - \mathbf{t}^{1\mathsf{T}}\mathbf{x}_i \\ y_m - \mathbf{t}^{2\mathsf{T}}\mathbf{x}_i \\ w_m - \mathbf{t}^{3\mathsf{T}}\mathbf{x}_i \end{pmatrix} = \mathbf{0}$$

$$\begin{pmatrix} x_m - t_{11}x_i - t_{12}y_i - t_{13}w_i \\ y_m - t_{21}x_i - t_{22}y_i - t_{23}w_i \\ w_m - w_i \end{pmatrix} = \mathbf{0}$$

This formula can be rewritten so that it is a linear equation system in terms of the parameters that should be found.

$$\underbrace{\begin{bmatrix} -x_i & -y_i & -w_i & 0 & 0 & 0 & x_m \\ 0 & 0 & 0 & -x_i & -y_i & -w_i & y_m \\ 0 & 0 & 0 & 0 & 0 & 0 & w_m - w_i \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} t_{11} \\ t_{12} \\ t_{13} \\ t_{21} \\ t_{22} \\ t_{23} \\ 1 \end{pmatrix}}_{\mathbf{t}} = \mathbf{0}$$

So the final form of the equation system that has to be solved is

$$\mathbf{A}\mathbf{t} = \mathbf{0}$$

Note that the solution vector $\mathbf{t}$ has an entry that has to be 1. This can be easily achieved by finding a solution where the last value is arbitrary and then dividing the whole vector by the last value. That way the last entry will be 1 and the equation system is still fulfilled.

The equation system is at the moment highly underdetermined and there are probably many solutions which fulfill the equation system. This is because only one correspondence pair is used here. So additional points have to be used. For each point the matrix $\mathbf{A}$ has to be computed. Because the solution vector has to hold for all of these matrices, the equation system can be extended by just concatenating the matrices row wise. So the final equation system that has to be solved looks like followed.

$$\begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \\ \vdots \\ \mathbf{A}_n \end{bmatrix} \mathbf{t} = \mathbf{0}$$

In this equation system $\mathbf{A}_i$ is the $\mathbf{A}$ matrix for the i-th correspondence pair and n is number of pairs used. In the implementation four pairs are used. Note that by doing so the equation system becomes overdetermined. If there are no errors in obtaining the correspondences the equation system would have one solution. Unfortunately there there will always be noise and also the limited amount of resolution alone will already add some error which leads to the equation system having no solution. Therefor the system has to be approximated. This can be done using the singular value decomposition(SVD)[5, p. 585ff]. The eigenvector that belongs to the smallest eigenvalue is the best solution for the parameters.

To conclude the following steps have to be done for the calibration.

1. Find correspondence pairs
2. Compute the matrix $\mathbf{A}_i$ for each pair
3. Concatenate all $\mathbf{A}_1 \dots \mathbf{A}_n$ and apply the SVD
4. Find the eigenvector with the smallest eigenvalue (the last one)
5. Normalize the solution vector so that the last entry is one
6. Put the parameters into the transformation matrix

3.3.4. Results

To see how well the auto focus performs a reference image and some test images with different offsets were made. Four of the test images are

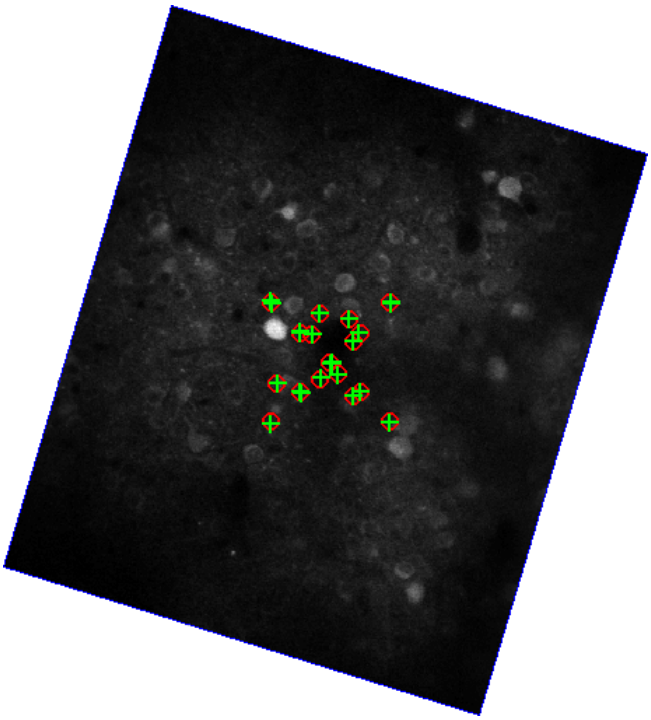

Figure 3.5.: Plots of the test results for the auto focus. The reference image transformed into the microscope coordinate system. Red circles mark the real center points of the test images and green crosses mark the calculated center points.

used for the calibration. Figure 3.5 shows the reference image transformed into the coordinate system of the microscope. The red circles are the real center points where the test images were made. The green crosses mark the computed center points. From this image it can be seen that the major transformation which has the most influence in the transformation between the coordinate systems is the rotation. Nevertheless some tests have shown that the other components are also very important to be able to find the correct transformation matrix. The plot also shows that the matches are quite good.

In Figure 3.6 the error that is made for each test image can be seen. It can be noticed that for all test images the error is lower than $2\mu m$ in each direction. The neurons that can be seen in the previous figure are usually in the range

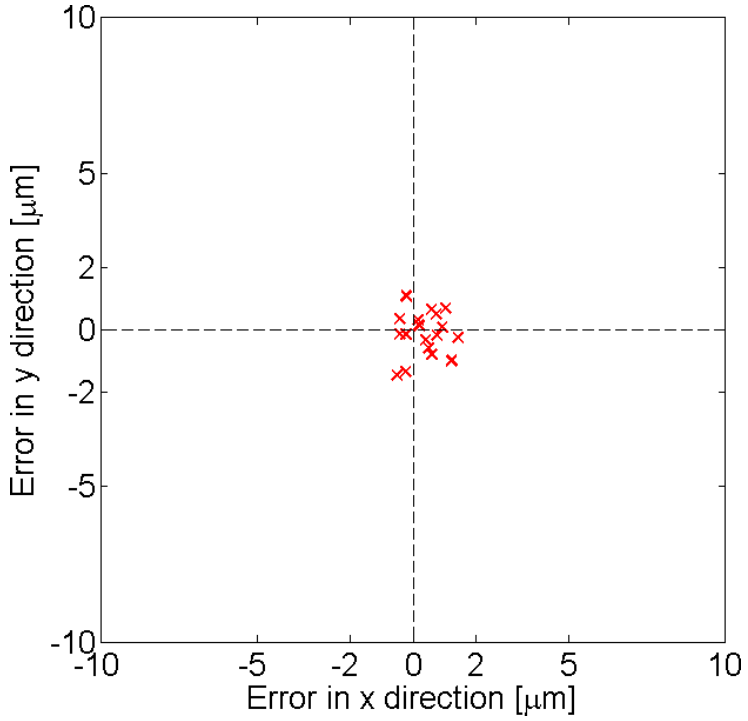

Figure 3.6.: Error between real center points and the calculated ones.

of $10 - 20\mu m$. This result is much better than trying to get to the same position by eye.

The auto focus is still in an early stage but the fundamental steps for further developments are taken. Tests have shown that the tools that are used for the implementation of the auto focus can theoretically produce relatively good results and have the potential to support the experimentalist. Unfortunately first test under real conditions have shown that there are still some flaws that have to be fixed.

Once the auto focus is at a state where it can be used without major problems one can think about how it can be extended to become even more useful. At the moment it is used to correctly get to a single position but with some extensions it would also be possible to define several regions where the auto focus should go to. This way it is possible to automatically make images

from different brain areas during the experiment which can lead to data that help to understand the interaction of these areas. It is also possible to extend the auto focus so that it does not just focus once at the beginning of the experiment but to do this after a fixed amount of time. This would allow the auto focus to correct for slow drifts of the brain. Additionally this could be combined with the network interface developed for this thesis so that the acquisition computer will not get slowed down. These are just a few examples that should show what can be done with this tool but already without these extensions it is already useful for the experimentalist and also for the images that are produced.

4. Conclusion

The primary goal of developing a network interface for ScanImage was achieved. By implementing and testing the interface it was shown that it is possible to send the high-resolution images fast enough over the network. To send the images over the network it is necessary to split the data into smaller data pieces at the sender and reassemble them at the receiver. By the benchmark tests it was shown that the reassembly of the images is fast enough that there is still time for analyses even if the data are sent at about $30Hz$. To send the data over the network UDP is used. This protocol has some disadvantages over the other possible choice namely TCP but has much less overhead and is therefor faster. The reordering of the packets is done by the implementation of the interface but it is still possible that some packets are lost. Fortunately the tests showed that the packet loss is negligible as long as the analyses are fast enough. Off course if the analyses are slower then the acquisition of the images some data are discarded because at some point the buffer that will receive the images is filled up.

The network interface implemented for this thesis can be used for many different applications. Some examples have been shown here but there are many other ways in which the interface can be used. Especially for application that should run in real time the network interface is very useful because it allows the application to run on a different computer which is not busy acquiring the data. This way the acquisition and the analyses can run in parallel and they will not influence each other as much as if the analyses are directly implemented as plugin for ScanImage. Therefor if the application that analyses the data crashes it will not influence the acquisition of the images. The biggest strength of this interface is that it defines exactly how the data have to be encoded when they are sent over the network. This makes the interface very flexible. For example it is possible to implement the receiver in any programming language that is able to work

with network connections. Therefor the application which is used to analyse the data can be implemented in any arbitrary language as long as there is an implementation of the receiver in the same language. Although it has a lot of advantages the interface is designed for local networks. Internet connections are usually much slower then connection in local networks therefor it is not possible to use the interface without problems for large networks. Additionally the latency plays a big role in large networks. The latency is the time that is needed to send the data from the sending computer to the receiving computer. In small networks it can be neglected but in larger networks this might cause more packets to be lost.

The interface should be seen as a prototype that should show that it is possible to send data over the network and process them in real time. There are many ideas in which way the interface can be improved or extended. The receiver is at the moment implemented in MATLAB. This has one major disadvantage namely MATLAB only runs in a single thread. Because of that it is not possible to assemble incoming data while the analyses runs. One could implement the server in C and compile it to a mex-file which would allow to do the assembly of the images on a different thread. This might increase the time that can be used for the analyses of the data. Another possible improvement is to generalize the meta data that are sent. At the moment the whole meta data of ScanImage are sent. A lot of these data are actually not needed and it forces the use of ScanImage for this interface. By generalizing the meta data it could even be possible to exchange the sender and use different acquisition software as long as it offers a way to send the images as they are acquired. A very interesting extension of the sender would be to add multicasting. This allows the sender to broadcast the data to several receiver which are interested in them. By doing so different computers can be used for different analyses. Note that only the generalization of the meta data would introduce a change in the interface. The other two ideas only require changes in the sender and the receiver can be used as before.

To show that the interface also works for real experiments additionally some application were implemented. A very useful one is the real-time tracer which allows to read out the activity of neurons from the images received over the network interface. This application was developed in collaboration with Mario Prsa. The real-time tracer can be used as a starting point for

other application that base their analyses on the activity of the neurons. On top of this the robot arm control was developed. It shows that it is possible to translate the images received using the network interface into physical movements. The third application that was developed was the auto focus. In the beginning it was also intended to use the network interface but it turned out that it is easier and more user-friendly when it is implemented directly as a plugin for ScanImage. The auto focus is still a work in progress project and some work has to be done to make it usable in the experiments but the core functionality works and is already tested with success. The benchmark tests together with the applications developed should show that the interface is a working and useful tool for experimental setups.

Appendix A.

Manual

This section covers how the interface for the fast read-out can be used. First it is explained what has to be done to install it and how ScanImage has to be configured. At the end it is shown how it can be used to analyse data. It is assumed that ScanImage and the microscope are already working. The files that are needed can be found on the CD which comes with this thesis.

A.1. Preparations

Before you start installing the streaming interface make sure that ScanImage is working properly. Then find the following two zip-files on the CD

- `interface.zip`
- `tcp_upd_ip_2.0.6.zip`

and extract them to different folders where you can find them. The first one contains all the necessary files for the interface. The second one contains the TCP/UDP/IP Toolbox 2.0.6[13]. It is a free implementation that allows to send packets, using TCP or UDP, over the network in MATLAB. In order to use the toolbox a file has to be compiled. Therefor the `mex` command in MATLAB has to be setup. To do so type `mex -setup` into the MATLAB command line. It will then asked if installed compilers should be located. Just hit enter to let MATLAB find all the compilers you have installed. Depending on the platform you are using different compilers will show up. When you are using Windows 32bit at least a compiler called LCC should be

in the list and Unix user should see the GCC. But if you are using Windows 64bit you will first have to download a compiler. You can find one on the support page[11] of MATLAB. Just click the button that says "Microsoft Windows SDK 7.1" and start the download. When it is done install the compiler by double clicking on the downloaded file. After installing it run `mex -setup` and a Microsoft Visual C++ compiler should show up in the list. To continue the setup of the `mex` command enter the number next to one of the compilers and press enter. It will then ask to verify your choice of the compiler. With pressing enter the setup will be finished. A more detailed explanation of the `mex` command can be found on the MATLAB help page[10].

A.2. Installation

When the preparations are done you can start to install the TCP/UDP/IP Toolbox. To do so open MATLAB and change the directory to the folder where you extracted the zip-file of the toolbox to. It should contain a folder called `tcp_udp_ip`. Inside this folder there are several files one is called `pnet.c`. Replace the file with the one you can find in the folder where you extracted `interface.zip` to. After that the file has to be compiled. Depending on the compiler you are using you have to enter the following command

- LCC
  `mex -O pnet.c {MATLAB_INSTALL_DIR}\sys\lcc\lib\wsock32.lib -DWIN32`
- Microsoft Visual C++
  `mex -O pnet.c ws2_32.lib -DWIN32`
- GCC
  `mex -O pnet.c`

Where `{MATLAB_INSTALL_DIR}` is the path to the directory where you installed MATLAB. When the compilation has finished you just have to add the `tcp_upd_ip` folder to your MATLAB paths. This can be done in the menu when you press File→Set Path.... A window will show up where you can find a button that says "Add Folder...". Press it and chose the `tcp_udp_ip` folder.

The installation of the interface is easier then the installation of the TCP/UDP/IP Toolbox. Like in the last step before you just have to add the folder with the contents of `interface.zip` to the MATLAB paths. Do so within the same menu as before.

A.3. Configuration

The next thing to do is to tell ScanImage that it should use the sender as a plugin. First you need to start ScanImage. When it is ready look for the window with the title "UserFunctions". If you do not see it you can open it from the main menu by selecting View→UserFunctions. In this window you can define which functions should be called for a specific event. Add the function `udpStreamingPlugin` to the following events

- `appOpen`
- `appClose`
- `frameAcquired`
- `acquisitionStarted`
- `acquisitionDone`

For the event `appOpen` you also need to add the IP address of the computer where the receiver is running as arguments to this method. Just write the IP address in curly brackets into the argument field of the `appOpen` event. Save the settings in a usr-file, close ScanImage and start it again. When it asks for user settings at start up choose the before saved file. The plugin will now send the images over the network.

A.4. Reading out images

The last thing to do is to define a function which will read the images from the receiver and perform some analysis on them. The receiver uses the event interface of MATLAB to call the function every time a frame is ready. In order that the receiver calls your function correctly its header has to have a specific format which looks like this:

```
function <name_of_function>(eventSrc, eventData)
```

where <name_of_function> should be replaced by the name of your function. If the function is ready you have to create the receiver, register your function, so that it gets called when a frame is ready, and then start the receiver. This can be done with the following lines of code:

```
server = UDPServer('4242');
server.addlistener('frameComplete', @<name_of_function>);
server.run;
```

The first line creates the receiver called UDPServer. The receiver needs one argument which is the port on which the data are sent. At the moment the port which is used to send the data by the sender is 4242, therefor you will also tell the receiver it should use this port. The next line tells the receiver it should call your function when an image is ready. You could also define more than one function, then you just have to repeat this line for each of them. Finally the receiver is started with the last line. It will run until a QUIT packet is received from the sender which will be sent when ScanImage is closed. You can also force close the receiver by pressing Ctrl-C. It may take up to 10 seconds until it closes when you use this way.

A.5. Examples

In this section some code examples are shown. They should show how to write code that uses the interface. These examples from this section can also be found on the CD accompanied with this thesis.

The first example shows how the image can be read out using the interface. The code in Listing A.1 uses imshow to plot each image as soon as it is received. The image that has been received is passed to the function in form of the variable eventData.image. The command drawnow in line 4 makes sure that the image is displayed on the screen every time the function is called. In Listing A.2 it is shown how the receiver can be started. First a new object which represents the receiver is created then the function from

Listing A.1 is added as callback function for the `frameComplete` event. This way it will be called every time a new image is received. Finally the receiver is started at the last line using the method `run`.

```
1 function processFrameImshow( eventSrc, eventData )
2
3 imshow(eventData.image, []);
4 drawnow;
5
6 end
```

Listing A.1: imshowEx.m

```
1 clear all;
2 close all;
3
4 server = UDPServer('4242');
5 server.addlistener('frameComplete',@imshowEx);
6 server.run;
```

Listing A.2: startImshow.m

Very often it is necessary to have variables which store some information that is updated every time the analysing function is called. This can be achieved in two ways. The first one is to use global variables. MATLAB allows you to define variables which can be accessed from everywhere in the current MATLAB session. Listing A.3 shows an example where the variable `counter` is increased every time a new image is received. If the variable would be declared in a normal way it is cleared as soon the function ends and the information of how many images have been received are lost. By defining `counter` as a global variable, as shown in line 3, the information will be stored until the variable is explicitly cleared. To make sure that the variable can be used correctly it has to be initialized, which is shown in Listing A.4. Again the variable is declared as global and is then set to zero. If this part is omitted the `counter` would be by default initialized as an empty matrix and the code from the previous listing would exit with an

error.

```
1 function processFrameImshow( eventSrc, eventData )
2
3 global counter;
4
5 counter = counter + 1;
6
7 end
```

Listing A.3: counterEx.m

```
1  clear all;
2  close all;
3
4  global counter;
5
6  counter = 0;
7
8  server = UDPServer('4242');
9  server.addlistener('frameComplete',@counterEx);
10 server.run;
```

Listing A.4: startCounterEx.m

Global variables are an easy way of storing information over several calls of the analysing function but for implementations where many of these variables are needed the code can get very messy. Fortunately the same goal can be achieved using classes. Listing A.5 shows the code which does the same as the code above but uses a class instead of a global variable to store the counter. A class consists of properties, which are variables that represent the state of the class and store information, and methods, which are functions that can access the properties and perform actions depending on them. In the example there are two properties. The variable `counter` is again used to store how many images have been received. The variable `server` stores the object which represents the receiver. The first method in

the code is the constructor. It will be called every time a new instance of this class is created which can be done as shown in Listing A.6 on line 4. So if a new instance is created the counter will be initialized to zero. The next method starts the server. In the previous examples this was done in the start script but by creating the receiver inside the class the object can be saved and the class has access to the methods of the receiver which allows for example to stop the receiver if anything unwanted happens. The last method is the one that is used as callback function for the receiver. Note that the function header here looks a bit different then in the previous examples. This is because the function is now a method and every method, except the constructor, has the reference to the current instance as the first parameter. The second and the third parameter are the same as before. What is also important is that the class has to be derived from the `handle` class which can be seen in the first line. This has to be done to be able to access the properties from inside the methods. The code in Listing A.6 creates an instance of `ClassesEx` and then starts it.

```
1  classdef ClassesEx < handle
2
3    properties
4      counter;
5      server;
6    end
7
8    methods
9
10     function obj = ClassesEx()
11       obj.counter = 0;
12     end
13
14     function start(obj)
15       obj.server = UDPServer('4242');
16       obj.server.addlistener('frameComplete',@obj.countUp);
17       obj.server.run;
18     end
19
```

```
20      function countUp(obj, eventSrc, eventData)
21        obj.counter = obj.counter + 1;
22      end
23
24    end
25
26  end
```

Listing A.5: ClassesEx.m

```
1  clear all;
2  close all;
3
4  exampleClass = ClassesEx();
5  exampleClass.start;
```

Listing A.6: startClassesEx.m

These are the basic concepts of how the interface can be used in the code. For a cleaner implementation the last method is recommended if variables are needed which keep information over several calls of the analysing function. If such variables are not needed the first method is the simplest and the fastest to implement. For quick tests and if only few persistent variables are needed the second method can be used as it is a bit faster to implement.

Bibliography

[1] A. Bosch, A. Zisserman, and X. Muoz. "Image Classification using Random Forests and Ferns." In: *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. 2007, pp. 1–8. DOI: 10.1109/ICCV.2007.4409066.

[2] W Denk, JH Strickler, and WW Webb. "Two-photon laser scanning fluorescence microscopy." In: *Science* 248.4951 (1990), pp. 73–76. DOI: 10.1126/science.2321027. eprint: http://www.sciencemag.org/content/248/4951/73.full.pdf. URL: http://www.sciencemag.org/content/248/4951/73.abstract.

[3] Li Fei-Fei, Rob Fergus, and Pietro Perona. "Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories." In: *Computer Vision and Image Understanding* 106.1 (2007). Special issue on Generative Model Based Vision, pp. 59 –70. ISSN: 1077-3142. DOI: http://dx.doi.org/10.1016/j.cviu.2005.09.012. URL: http://www.sciencedirect.com/science/article/pii/S1077314206001688.

[4] R. Fergus, P. Perona, and A. Zisserman. "Object class recognition by unsupervised scale-invariant learning." In: *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*. Vol. 2. 2003, DOI: 10.1109/CVPR.2003.1211479.

[5] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Second. Cambridge University Press, ISBN: 0521540518, 2004.

[6] Vijay Iyer. *ScanImage*. 2013. URL: http://www.scanimage.org/.

[7] Sung Hyun Jo et al. "Nanoscale Memristor Device as Synapse in Neuromorphic Systems." In: *Nano Letters* 10.4 (2010). PMID: 20192230, pp. 1297–1301. DOI: 10.1021/nl904092h. eprint: http://pubs.acs.org/doi/pdf/10.1021/nl904092h. URL: http://pubs.acs.org/doi/abs/10.1021/nl904092h.

[8] JPl Lewis. "Fast template matching." In: *Vision Interface*. Vol. 95. 120123. 1995, pp. 15–19.

[9] Loren L Looger and Oliver Griesbeck. "Genetically encoded neural activity indicators." In: *Current Opinion in Neurobiology* 22.1 (2012). Neurotechnology, pp. 18 –23. ISSN: 0959-4388. DOI: http://dx.doi.org/10.1016/j.conb.2011.10.024. URL: http://www.sciencedirect.com/science/article/pii/S0959438811001917.

[10] Mathworks. *Build MEX-Files*. 2013. URL: http://www.mathworks.de/de/help/matlab/matlab_external/building-mex-files.html.

[11] Mathworks. *Supported and Compatible Compilers – Release 2013a*. 2013. URL: http://www.mathworks.de/support/compilers/R2013a/index.html.

[12] T. A. Pologruto, B. L. Sabatini, and K. Svoboda. "ScanImage: flexible software for operating laser scanning microscopes." In: *Biomed Eng Online* 2 (May 2003), p. 13.

[13] Peter Rydesäter. *TCP/UDP/IP Toolbox 2.0.6*. 2001. URL: http://www.mathworks.com/matlabcentral/fileexchange/345-tcpudpip-toolbox-2-0-6.

[14] Christoph Stosiek et al. "In vivo two-photon calcium imaging of neuronal networks." In: *Proceedings of the National Academy of Sciences* 100.12 (2003), pp. 7319–7324. DOI: 10.1073/pnas.1232232100. eprint: http://www.pnas.org/content/100/12/7319.full.pdf+html. URL: http://www.pnas.org/content/100/12/7319.abstract.

[15] Karel Svoboda and Ryohei Yasuda. "Principles of Two-Photon Excitation Microscopy and Its Applications to Neuroscience." In: *Neuron* 50.6 (2006), pp. 823 –839. ISSN: 0896-6273. DOI: http://dx.doi.org/10.1016/j.neuron.2006.05.019. URL: http://www.sciencedirect.com/science/article/pii/S0896627306004119.

[16] Carlos Zamarreño-Ramos et al. “On spike-timing-dependent-plasticity, memristive devices, and building a self-learning visual cortex.” In: *Frontiers in neuroscience* 5 (2011).