Appendix

Introduction

Functional imaging tools have greatly contributed to our understanding of many biological systems. The development of dyes, which are sensitive to voltage changes, changes in the calcium concentration, pH or glutamate and the possibility to genetically express them in selected cell populations allow for insights into physiological processes undreamt of until recently. One drawback, though, is the large amount of data generated. The evaluation of this data is a painstakingly time consuming process, which often takes much longer than the actual experiments. Additionally, in most of the cases it is only a necessary step before the real data analysis and interpretation can start. I have developed a computer program, which enhances the speed of imaging data evaluation and facilitates this process. This increased evaluation speed enables scientists to record and evaluate more data or to spend more time on the final data analysis.

Starting point

Several competing computer programs for bio imaging exist. While many programs are exclusively sold along with the imaging hardware, others like Metamorph (Universal Imaging Corporation) can be acquired separately. These programs offer a wide range of features, ranging from device drivers for scan tables and CCD cameras to actual image analysis and even image deconvolution. These features make the programs very versatile, but also expensive. Additionally, no commercially available product contains all features necessary for the analysis of imaging data based on odor evoked glomerular activity. Therefore in our laboratory data evaluation was done with a series of self written scripts, which I will subsequently call the *View* scripts.

View scripts

These scripts are implemented using the programming language IDL (RSI, Boulder, CO). They originate from the late nineties, when calcium imaging was established in the institute by Jasdan Joerges and others. Further on they were maintained

and extended by Giovanni Galizia. Growing over time, these scripts have proven to be extremely flexible and powerful, allowing for data evaluation and manipulation in many different ways. One major drawback, though, is that a lot of programming expertise is needed to use them and that data evaluation with the *View* scripts is time consuming.

Subsequently, I will outline the different steps necessary for data analysis with the *View* scripts:

1) Data acquisition using *TILLVision* software

The term data acquisition describes the actual recording of the measurements. In our laboratory this is done using the *TILLVision* software (Till Photonics; Martinsried, Germany). In addition to controlling different devices, like the monochromator and the shutter, a protocol editor allows for the adjustment of the different experimental parameters. *TILLVison* saves the measurements grabbed by the CCD camera to the hard disk. Measurements from each experiment (animal) are saved in the form of a vision workspace file *(*.vws*) containing most of the experimental parameters and a folder (**.pst*) which contains the actual recordings.

2) Generation of a *log* file from the *TILLVision* workspace

The *TILLVision* workspace is saved in proprietary format whose specifications are unknown. To access the different experimental parameters for the subsequent data analysis, the information contained in the **.vws* files has to be transferred to text files. This can be done from within *TILLVision*, using a macro. The resulting text file is called *log* file. *log* files contain the list of measurements recorded from one animal, together with the recording parameters. Among others, these parameters include the names of the measurements, their pixel sizes, the wavelengths used for the recordings and the time point at which the single frames were recorded, saved in Coordinated Universal Time (UTC).

3) Generation of a *list* file from the *log* file

The above mentioned *log* files still do not contain all information necessary for further data evaluation. For example the on- and offset and the identity of the odor stimulus are unknown to *TILLVision* and therefore have to be added manually. As the structure of the

log files makes them unsuited for efficient manual editing, the information contained in *log* file has to be transferred into another tab separated text file, the so called *list* file. This is done using an IDL routine, called *log2list_block.pro*. With some preliminary knowledge of IDL programming, and more sophisticated knowledge of the script itself, recurrent features like stimulus on and offset can be encoded in the *log2list_block.pro* routine. This reduces the amount of manual editing necessary afterwards.

4) Manually editing the *list* file

Some information, like the used odors or form of the treatment in cases where pharmacological reagents or a sucrose reward were applied, must be entered manually into the *list* files. The tab delimited nature of the *list* files facilitates such editing in spreadsheet programs like Microsoft Excel.

5) Creation of *GR_* and *master* programs

With all information about the measurements accessible via the *list files*, the *View* scripts now have to be set to evaluate the data. This is done by creating so called *GR_* programs, one for each experiment to evaluate. This process is rather fast, as conveniently enough, the list file created by *log2list_block.pro*, contains all necessary information. This is copied into an empty file and saved. Then the *master* program has to be created. In the master program the parameters for the data analysis have to be set. These parameters include the location of the data on the harddrive, the filter settings, how signals have to be calculated (for example for ratiometric data), and the output format of the data (e.g. as false color coded images, avi files, or as tif stack for shift correction).

Once the *GR_* and *master* programs are generated, further data analysis consists of repetitive execution of the master program with changed parameters for data output.

6) Shift correction

Since animals often move during the recording period different measurements obtained from the same animal have to be aligned. This is done using the shift correction. For the shift correction the output parameter in the *master* file is set to return a tiff stack for each

experiment containing one raw fluorescence image per measurement. These stacks can manually aligned one by one, using a script called *CompareSlicesAndShiftThem.pro*. After manual alignment, this script returns a tab delimited text file containing two columns of integers representing the X and Y shifts for each measurement. These can be copied into the *list* file and subsequent executions of the master program, with the according parameter for the shift correction set, will result in corrected data.

7) Region of interest (ROI) selection

There are two possible ways of selecting ROIs: View, a front end of the *View* scripts, supplies the user with a graphical user interface (GUI). With this user interface, single measurements can be loaded and a false color coded image is created. Mouse movements on top of the image result in on line traces, showing the fluorescence values of the underlying pixel over time, as well as the pixel position in X and Y. These pixel positions have to be manually entered into another tab delimited text file, the **.coor* file.

The second possibility to select morphological structures in the data is based on afterstainings. These additional stainings are applied after the measurement and delineate the glomerular borders. These borders can be extracted using image editing programs like Adobe Photoshop and are saved in a *.tif file

8) Traces generation

After setting the parameters in the master file for the location of the ROI files and the appropriate output type, the *master* program is executed a last time. Now for each animal and measurement listed in the *master* program and for each ROI, the pixel values over time (traces) are calculated and saved into one tab delimited text file, called **.gloDataMix*, per animal. The *gloDataMix* files are the basis for all quantitative and statistical analyses.

Development goals

As mentioned before, the *View* Scripts are extremely powerful and flexible. Nevertheless they have several limitations, like the lengthy evaluation process and, even

more important, the high degree of expertise necessary for a proper use. Another limitation is of a more historical nature. The development of the *View* scripts started in the nineties. Then the computers had less processing power and memory, but at the same time the CCD chips of the recording cameras already had a reasonable number of pixels. One measurement, recorded at the lowest resolution could already have a size of half a megabyte, while an expansive workstation would have 16 megabytes of main memory. Therefore the Scripts were designed to evaluate the measurements one after the other, which made online comparisons between measurements impossible. The enormous increase in processor power and main memory size during the recent years now allows for all measurements of several experiments to be at the same time in memory. This implicates a profound change in the program design which was not feasible via small changes in the *View* Scripts. Therefore, instead of modifying them, I decided to write an entirely new program.

Subsequently I will line out the development goals for this program and the strategies applied to achieve them:

Ease of use

The program should be easy to use, even without much knowledge of the underlying computing steps. Therefore I designed a user interface which can be completely operated by mouse.

Compatibility with the *View* scripts

The *View* scripts contain a large number of well written routines. In order to make use of some of them, I implemented the program in IDL. This also allows for a combination of both the *View* scripts and the new program during data evaluation. Additionally, the results of all intermediate steps of data evaluation, for example the movement correction or region of interest (ROI) selection, can be saved and later be used in the *View* scripts.

Correction of movement within measurements

In the *View* scripts, only shifts between measurements could be corrected. The new program should also allow for the correction of movement within measurements. Therefore I implemented an algorithm for automatic movement correction. Additionally, I based the movement correction window on the *CompareSlicesAndShiftThem.pro* and added some features to further facilitate its use.

Online comparison between measurements

The program should allow for many different measurements of different experiments to be online at the same time. The fundament of the new program was to be based on one large array, containing an arbitrary number of measurements. An accompanying list should contain all experimental parameters. Only the size of the main memory should limit the number of measurements to be visualized and evaluated at the same time.

Glomerulus identification

The program should facilitate glomerulus identification. The regions of interest (ROIs), which mark the position of a glomerulus, should be easy to choose. In the program this should be accomplished by simple mouse button clicks.

Speed of data evaluation

The program should speed up the overall evaluation time. This should be achieved by a combination of strategies. The generation of *list* files, *GR_* and *master* programs, as described above, should not be needed any more. Additionally all processing steps, like the movement correction, glomerulus identification and traces generation should be facilitated and done online, without reloading the data for each step.

The structure of the program

IDL

In order guarantee the interoperability between the program and the IDL scripts, I decided to implement the program in IDL. IDL (Interactive data language) is a high level computing environment. It consists of an integrated development environment (IDE), a complete array-oriented programming language and large amount of routines and libraries for data analysis, graphical presentation and statistical evaluation.

The IDL language, like other high level computer languages like Java or Python, is interpreted. This means that upon compilation, IDL code is not directly transferred into machine code, but into an intermediate code. A run time compiler (RTC) translates this intermediate code into machine code during the execution of the program. In the case of IDL, this has several advantages. Variable declaration is not necessary for most situations. Also the memory management is fully under control of the RTC, thereby preventing buffer overflows. A so called garbage collector constantly scans the computer memory for variables which have become obsolete which it deletes. This prevents memory leaks, a common cause for program or even operating system instability.

Another advantage of IDL and interpreted languages in general is that programming is fast. The syntax is easy to understand and write. A function written in IDL has generally one quarter of the number of lines as the same function written in a lower level language like C or C++ would have. This increases the programming speed considerably. Additionally, the first compilation step is very fast. Therefore new functions or routines can be tested immediately.

The disadvantage is that programs written in interpreted languages tend to be slower than their machine code counterparts. The RTC translates the program into machine code during run time and this translation needs time. In recent years, due to the ever faster computer processors and faster RTCs, program execution speed has ceased to be an issue for most applications. In IDL, calculation intensive routines are written in C and linked into the applications as libraries. The result is that the execution time of IDL code is fast enough for most applications, while its development time is very short.

The user interface

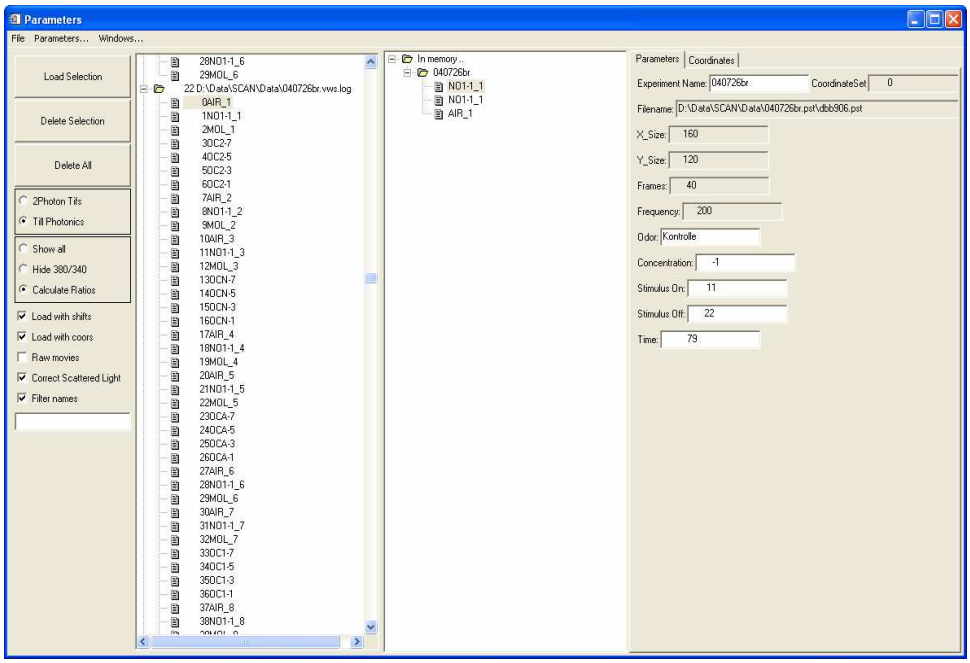
The user interface consists of seven different windows. Each window was specifically designed to fulfill one or several of the development goals. I will briefly

introduce the different windows. To remain concise, I will only explain their most prominent functions and leave out those functions, which are not important for a general understanding of how the program manages the evaluation of imaging data.

Parameters window

The Parameters window is the main window of the program. Through a dropdown menu in this window, all other windows can be accessed. Additionally it fulfils several other functions. *log* files created by *TILLVision* can be read and all measurements contained therein can directly be loaded in this window. The direct access to data makes the generation of *list* files, as well as *master* and *GR_* programs obsolete. Data can be evaluated as soon as the *log* file is generated, e.g. within seconds after measuring the data. Loadable and loaded measurements are shown in two different tree structures. Selected measurements can be loaded, or in case they are already in memory, deleted. The parameters of loaded measurements can also be edited in this window. These changes can be saved, tagged to the measurements, and are automatically loaded whenever the measurements are loaded again.

In addition to the *TILLVision* imaging data, the program now also supports image series in the tif format, an export method supported by all confocal and 2-photon microscopes.



A1) The parameters window

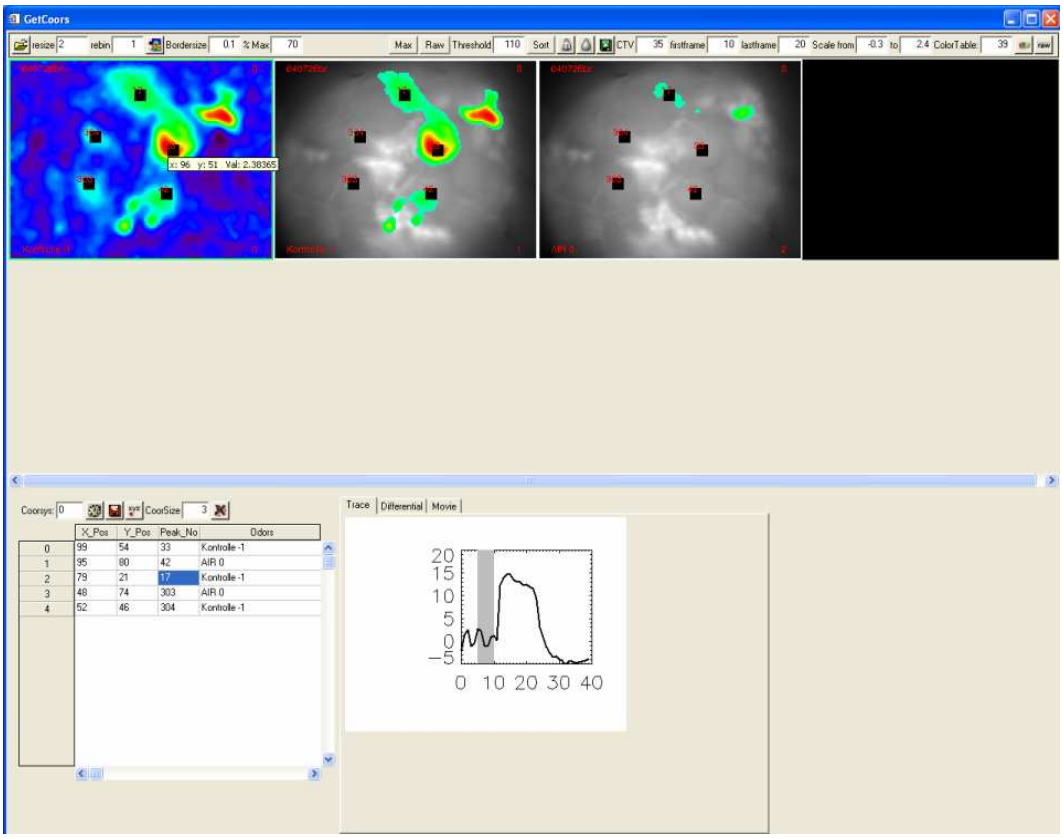
GetCoors window

The GetCoors window used to be the main window in previous program versions. Therefore it houses a great part of the functionality of the program. Its main task is the selection of different regions of interest (ROIs). Additionally, it provides different ways of data visualization. These visualizations can then be exported for later use in publications or presentations. Existing export methods are tiff images for the false color coded images as well as mpeg for movies of the measurements. All measurements contained in memory are shown as representative false color coded images in the GetCoors main window. The parameters according to which these images are calculated can be adjusted here. Single measurements can be watched as time series, and moving the mouse over the false color coded images returns the fluorescence value over time of the subjacent region. Pressing the left mouse button over such a region marks it as ROI. As these ROIs are considered to mark morphological features, such as olfactory glomeruli, they are transferred to all measurements from the same animal. Up to date ROI sets of up to 40 animals can be edited at the same time. In addition to manually selecting ROIs, a special algorithm allows for automatic selection of the active glomeruli.

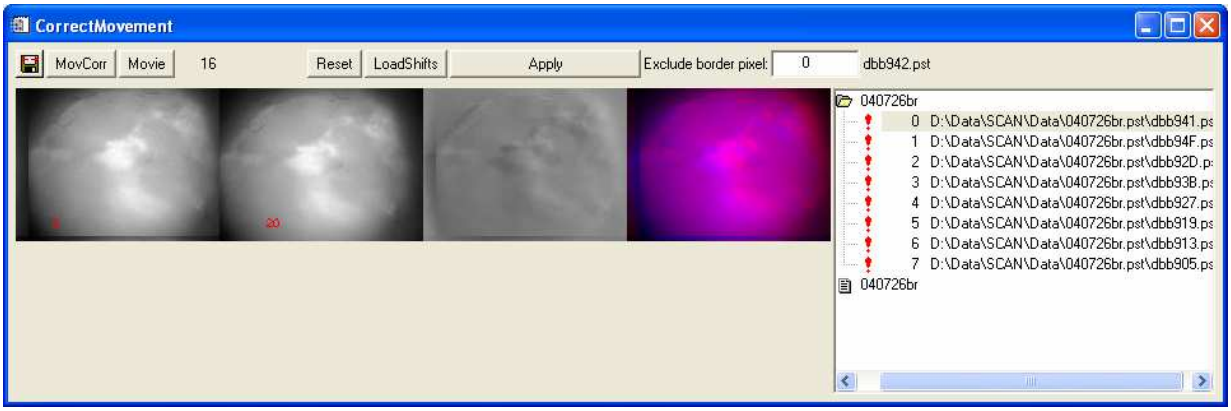
MovementCorrect window

The MovementCorrect window, as its name relates, mediates the steps necessary for movement correction. As the movement correction has to be done on morphological data, this window allows the loading of raw, unprocessed imaging data via a tree structure. The general way to do the movement correction is to first correct for movements within each measurement and then correct for shifts between measurements. All corrections for shifts between measurements are also directly applied to the false color coded images in the GetCoors window. An algorithm for automatic movement correction can be used to correct for movement both within and between measurements.

This algorithm will be explained in more detail in the algorithms section.



A2) The GetCoors window

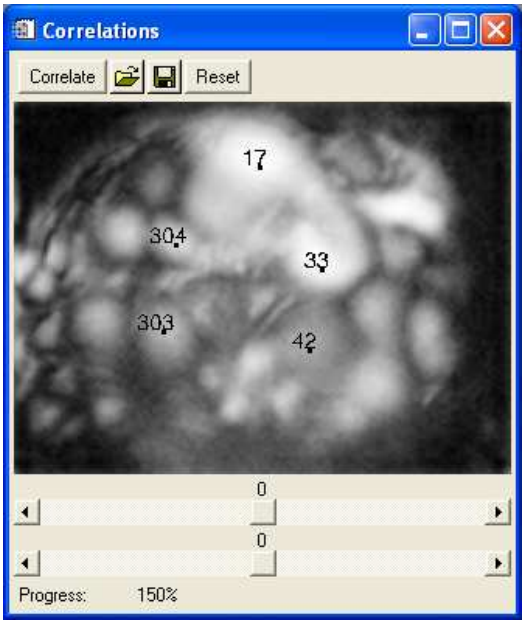


A3) The MovementCorrect window

Correlations window

The correlations window is the interface to a simple, but efficient algorithm. The result of the algorithm is an image outlining the physiologically active structures in the data, in our case of the glomeruli. When it comes to ROI selection, this window has a similar functionality as the GetCoors window. ROIs can be set, removed or moved on the window. Therefore this window is very helpful for the task of glomerulus identification.

The algorithm will be explained in more detail in the algorithms section.



A4) The Correlations window

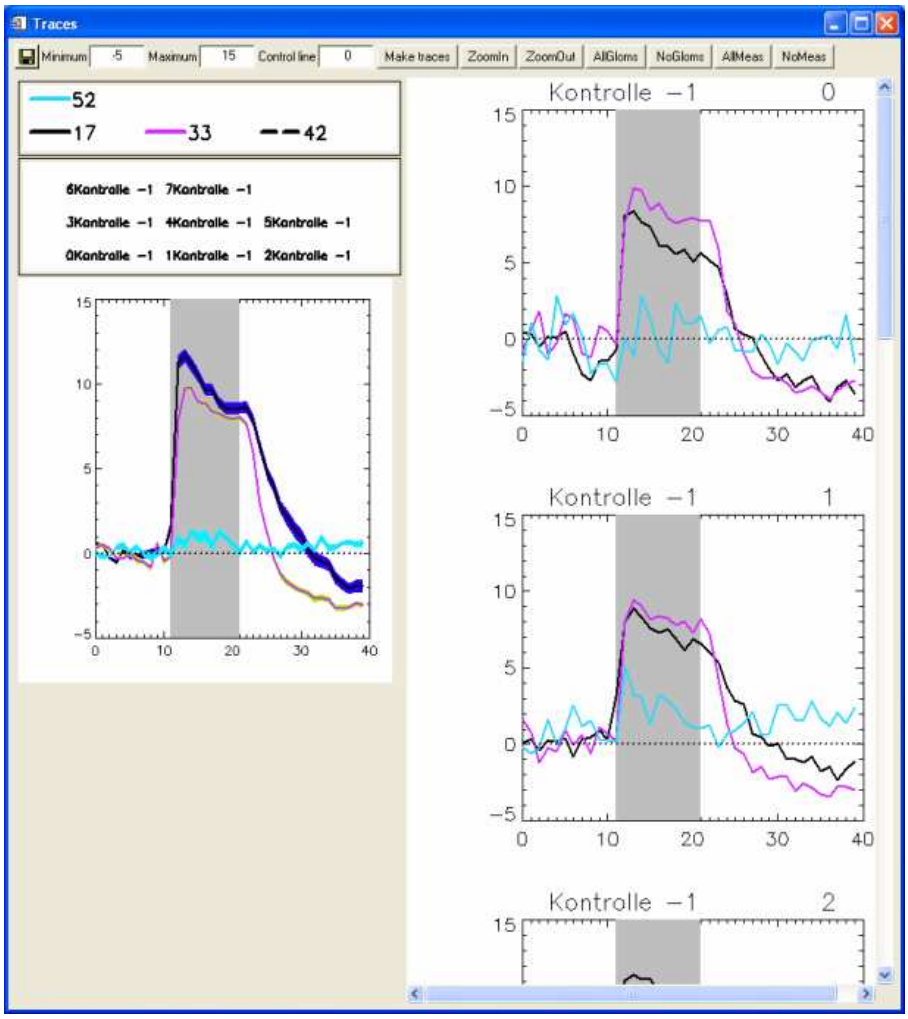
Traces window

In the Traces window, the change in fluorescence is plotted over time for each ROI in each measurement e. Different measurements and ROIs can be selected or deselected for plotting. As an additional feature, an extra graph shows the mean and standard errors of the ROIs, averaged across all selected data.

While one task of this window is to visualize the behavior of ROIs over time, the so called traces, its most important task is to allow for the export of these traces. Up to now, traces can be exported in two formats. One format is the so called *gloDataMix* format. This is the same format as generated by the *View* scripts described previously. It is a tab delimited text file which is readable by Microsoft Excel and for whose further evaluation many different IDL scripts exist. The second format is also text based, but here the data is arranged in a way which is ideal for loading it into relational databases like MySQL or PostGreSQL or statistical programs.

All graphs can be directly copied into the Windows clipboard and then pasted into other applications like Microsoft Powerpoint or Word. One drawback of this method is that all graphs are exported as bitmaps, which makes them unsuited fir further editing.

Therefore I have started to implement additional export functions, like a direct vector based PDF and WMF export of the graphs. These features will be included in future versions of the program.



A5) The traces window

Additional windows

Two other windows exist, which mediate special cases of calcium imaging analysis. One is allows for the generation of data masks, which can be overlaid over the data. The second is for glomerulus interpolation. As both are not of immediate importance for successful data evaluation, I will not explain them in further detail.

Algorithms

In this part I will introduce some of the algorithms used in the computer program. Most of the algorithms used in this program were not developed by me but were taken from libraries. These algorithms, like the digital filters, data sorting algorithms or data import and export routines, contribute greatly to the functionality and complexity of the program. Nevertheless, I will limit myself to introducing two of the algorithms, which at least partly were developed by myself. These algorithms are of special importance for the functionality of the program: A) the movement correction, B) the correlation algorithm.

Movement correction

I) Theoretical background

In vivo imaging has the advantage of delivering data from living animals. This can be considered as a big advantage, as the experimentators aim is to obtain his data under the most natural circumstances as possible. On the other hand several inconveniences come along. One of the biggest problems is that living animals move. As the regions of interest selected in imaging measurements are defined by their position in the images, these movement artifacts have to be corrected.

Correcting data for movements is a problem which can be separated into two parts. While the first part is the definition of an error function, the second part is finding the minimal error as defined by such an error function.

1) The error function

In the case of the movement correction, the task of the error function can be defined as follows: a) It has to return an error value for all possible shifts of one image relative to the other and b) this error value has to be minimal when the images are optimally aligned. A common choice for an error function is the sum of the pixel wise differences (SPD):

$$err = \sum \left| x_{ij} - y_{ij} \right|$$

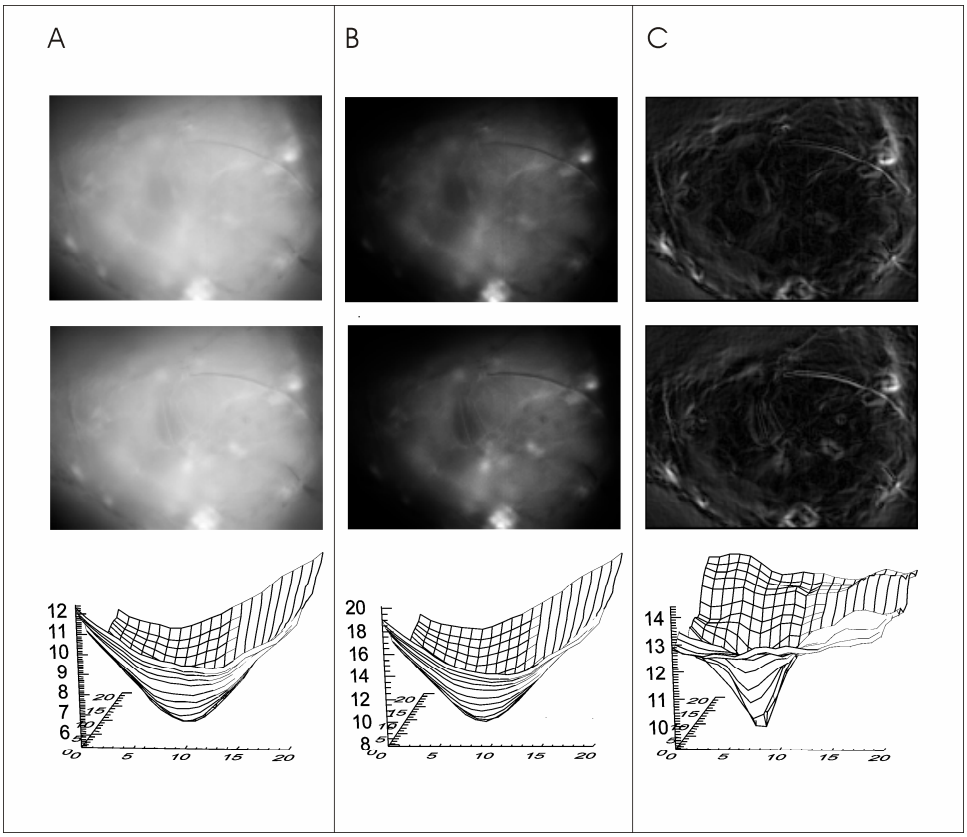
Where *err* is the error value and *i*, *j* depict the position of the pixel in the two images *x* and *y*. Imagine two identical images perfectly aligned on top of each other. In this case the SPD error will be zero. A shift of either image will result in an increase in the SPD error. The sum of the absolute pixel differences is chosen in order to avoid situations in which an increase in the difference between some pixels could be compensated by a decrease below zero in other pixels. Another common way to achieve this is to use the quadratic pixel wise difference:

$$err = \sum \left(x_{ij} - y_{ij} \right)^2$$

This error function has the additional advantage that the gradients within the error function increase.

By systematically shifting one image on top of the other and calculating the error value for each shift, the error function defines an error surface. Examples of such error surfaces can be seen in figure A6).

It is obvious, that the quality of the movement correction directly depends on whether the error function is at its unambiguous minimum when the images to be corrected are optimally aligned. This may not be the case when images are blurred or have little contrast, which was often the case in the raw data of measurements conducted with calcium green AM, for example. Most movements occur along all three spatial axes, also including movement in and out of focus, the images to be aligned can vary considerably. To enhance the performance of the error function, I tried several preprocessing steps. These preprocessing steps included algorithms for contrast enhancement and edge detection, which were applied to the images before calculating the



A6) Movement Correction. A) Sample images from two different measurements and the error surface calculated by systematically shifting one image on top of the other and calculating the absolute pixel wise difference. B) The same two images and the error surface after contrast enhancement. C) The two images and the error surface after applying a Sobel edge detection filter.

error values. For the contrast enhancement, each pixel in the image was treated as follows:

$$val_{ij} = val_{ij}^4$$

Where val_{ij} represents the pixel values. The operators for edge detection were the commonly used Sobel and Roberts filters. The Sobel filter can be approximated by convolving the images with the following masks:

$$Xmask = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad Ymask = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

The masks used for the Roberts filter are very similar:

$$Xmask = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \qquad Ymask = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Figure A6) shows the example of two raw images as well as the same images after contrast enhancement and edge detection. Below are the error surfaces of the three approaches as defined by the absolute pixel wise differences between the two images, shifted against each other for -10 to 10 pixels along both axes.

2) Finding the minimum defined by the error function

The second part of the movement correction procedure is an algorithm with the task of finding the minimum defined by the error function. Several standard algorithms can be used here and some will be explained in more detail.

a) The brute force approach

In this approach, one image is shifted on top of the other in all possible positions. After each shift, the error value is calculated and the result is stored in a temporary variable together with the shifts which led to this result. After all shifts have been accomplished and all error values have been calculated, the minimum error value depicts the best correction. This procedure has the advantage that it always finds the global minimum. The obvious disadvantage is that it finds it in a very inefficient way, by testing all possible combinations. This may still be feasible in cases where only a limited number of images have to be corrected for movement. In most cases the brute force method will be too slow to be practicable.

b) The gradient descent

The gradient descent only calculates the error values for the shifts to the eight neighboring pixels, then choosing the smallest one of them. This is done iteratively, until any further shift would lead to a higher error value, e.g. a minimum is found. The advantage of this algorithm is that it strongly reduces the processing time, as instead of

all possible, only a small and defined set of error values have to be calculated. The disadvantage is that it is sensitive to local minima and may remain there without finding the global minimum. Several enhancements to the gradient descent therefore have tried to circumvent this problem, with differing success.

c) Simulated annealing

The original algorithm, developed in 1953 by Nicolas Metropolis et al., is such an improved version of the gradient descent. As explained above, the problem of the gradient descent is that it exclusively steps down the error gradient, and therefore gets trapped in local minima. Simulated annealing includes a perturbation into the calculation, thereby endowing the algorithm with flexibility. The probability that such a perturbation occurs depends on the energy state of the system, which in our case is the found error value. While the error value is high, the probability for perturbations is high, too. After a predefined number of iterations the algorithm stops.

In the case of the movement correction, three parameters have to be defined. First, what form does this perturbation have and how strong is it? Second, how does the algorithm know how high the relative error value is, e.g. how high is the probability for the perturbation? And third, how many iterations have to be used?

i) I implemented the perturbation as a jump to a randomly chosen pixel on the rim of a square of 5x5 pixels, whose center defined the actual position.

ii) The probability for the occurrence of such a perturbation was set to:

$$probability_{pert} = e^{\frac{err_t - err_0}{\Delta err}}$$

Where Δerr is the change in the error value between iterations, err_0 is the error value at time point zero (before starting the correction) and err_t the actual error value. Mark that with an increasing value of $(err_t\text{-}err_0)/\Delta err$ the probability for a permutation to occur decreases.

iii) In the absence of knowledge about the global minimal error value, I decided for a number of 200 iterations

II) Implementation

Trying to find the optimal algorithm for an automatic movement correction, I implemented and compared the preprocessing steps and the algorithms for finding the minimum of the error function. As error function, I tested both the absolute and quadratic pixel wise errors described above.

Subsequently I will describe my observations and conclusions:

1) There was no difference in the performance of the automatic movement correction between the error functions using the quadratic and the absolute pixel wise difference.

2) None of the approaches I tested resulted in a performance of the algorithm to a extent that would allow for unsupervised movement correction. While the general performance was good for all approaches, further manual corrections were often necessary. The use of the Sobel edge detection algorithm resulted in error surfaces with a more defined, unique minimum, and in many cases performed slightly better than when the error function was calculated on the base of the contrast enhanced or the untreated images. Additionally it is a fast algorithm and did not significantly increase the time needed for movement correction. Therefore I included it as standard feature into the program.

3) As expected, the brute force algorithm for finding the minima had the best performance, but was rather slow. The gradient descent was much faster and in many cases worked optimally on the smooth error surfaces defined by the untreated and contrast enhanced images. Its performance with images to which the Sobel edge detection algorithm had been applied, in turn, was very bad. This was due to the ragged surfaces described by the error function, which had many local minima. Though the performance of the simulated annealing algorithm in this case was better, it was very slow, in some cases even slower than the brute force algorithm. I did not find out whether this was because the algorithm is not suited for the task of fast movement correction or whether my implementation was faulty. Also more precise calculation of the number of iterations to go through might have resulted in a faster performance.

As most of the movement artifacts to be corrected reflected shifts of less than 5 pixels, I decided to implement a truncated version of the brute force algorithm. Now the

algorithm did not calculate all possible shifts, but only those of a distance of up to 10 pixels. As a result, the algorithm was now fast enough to correct even large image stacks, without sacrificing much of its performance.

III) Result

As I did not succeed in implementing an algorithm for fully unsupervised movement correction, I included several features which facilitate the manual correction for movement. Together with the implemented algorithms, these features allow for a rather fast correction of movement.

Correlation algorithm

I) Theoretical background

The selection of regions of interest (ROIs) is one of the crucial steps in functional imaging. In some cases, like single cell imaging, this task is rather straightforward. This is not the case for AL or MB recordings, where the same dye stains several distinct structures, which have to be told apart. The task of the correlation algorithm is to deliver a template to help with the selection of ROIs. This algorithm helps especially in cases where the raw fluorescence does not allow for a differentiation of the underlying structures, for example when data was obtained using bath-applied Calcium green AM or GCamp stainings of Drosophila projection neurons.

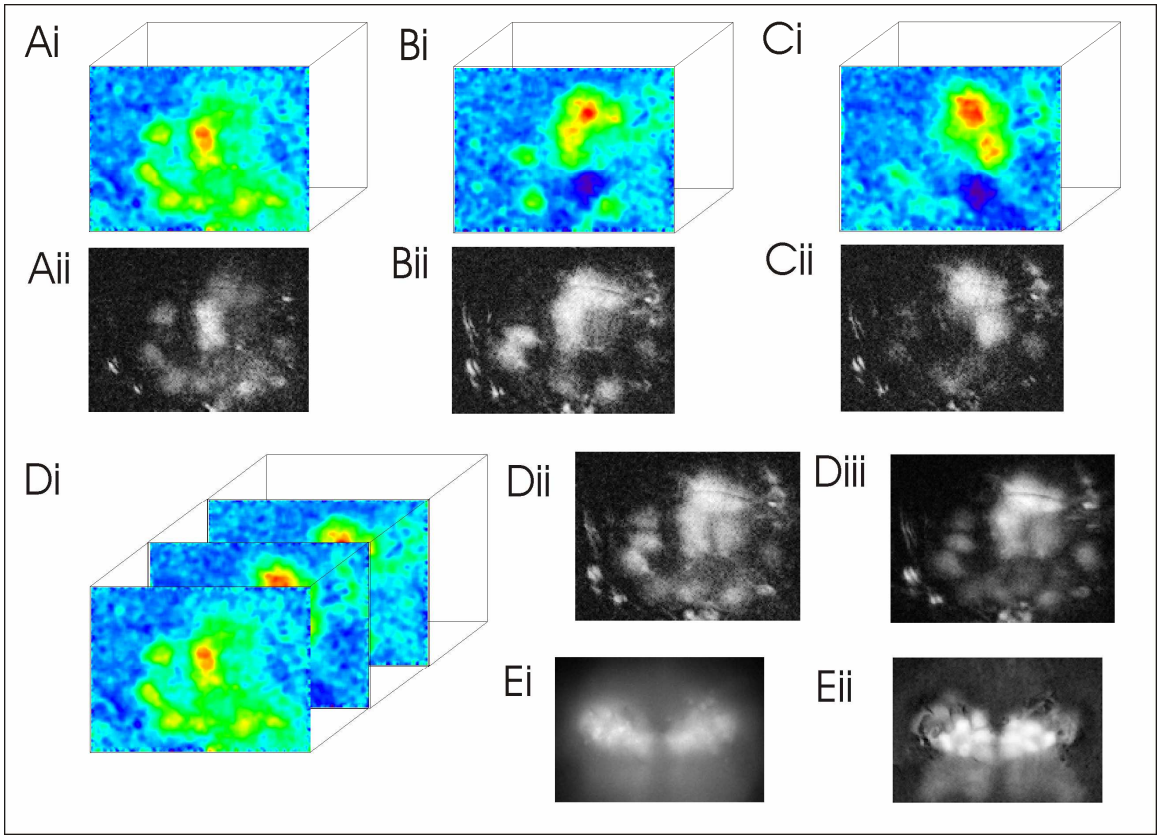
This procedure is not based on the raw fluorescence data but on the calculated signals, for example after calculating the ratios between the frames measured at two different wavelengths and subtracting the background fluorescence. The algorithm calculates the average degree of correlation between each pixel and its neighbors using the Pearson's moment product correlation coefficient. The rationale behind this is that glomeruli act as functional units (Wachowiak *et al.*, 2004). Therefore all pixels belonging to the same glomerulus should be active or inactive at the same time, resulting in high correlation values. Pixels belonging to different glomeruli in turn should be uncorrelated. It is obvious that this procedure can only work well on data which has been corrected for

movement, because only then do the pixel positions in different frames represent the same morphological structure.

For the algorithm the fluorescence values of a pixel at different time points (the frames of the measurement) constitute the elements of a vector. Then the correlation between the vectors of neighboring pixels x and y is calculated as follows:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Where n is the length and $\bar{x}, \bar{y}$ are the mean values of the two vectors x and y. The correlation coefficient may take any value between -1.0 and 1.0. A value of 1 shows that a linear equation describes the relationship between x and y perfectly and positively, with all data points lying on the same line and with y increasing with x. A score of −1 shows that all data points lie on a single line but that y increases as x decreases. A value of 0 shows that there is no linear relationship between the variables.



A7) Correlation algorithm. Ai-Ci) Schematic representation of three measurements recorded in the same animal, based on a low quality Fura-2 dextran staining. Aii-Cii) Resulting correlation image when applying the algorithm on each measurement separately. Di) Schematic concatenation of the three measurements. Dii) Correlation image of the three concatenated measurements. Diii) Correlation image after concatenating all 27 measurements recorded in this animal. Ei) Raw fluorescence image of the two antennal lobes in *Drosophila* with stained projection neurons (GH-146:Gal4-UAS:6GCamp). Eii) Resulting correlation image for this animal.

II) Implementation

For each pixel in a measurement, the correlation is calculated for each neighboring pixel and the resulting correlation values are averaged. The algorithm returns a correlation image in which all pixels have a value between -1 and 1. When applied to single measurements, the quality of the resulting images is often poor. This can be explained by the fact that changes in fluorescence in non active glomeruli are dominated by noise. As noise is always uncorrelated, pixels belonging to these glomeruli are uncorrelated, too. Therefore only those glomeruli which had been strongly active during the measurement can be discriminated in these correlation images.

To circumvent this problem the algorithm concatenates all measurements of responses to various odors obtained from the same animal as shown in Figure A7). The quality of the resulting correlation image is a direct result of the number of concatenated measurements and of the number of different odors measured.

III) Result

Though often helpful, this algorithm is only one of several tools in the program to facilitate the selection of ROIs. Other algorithms like the automatic selection of activity spots in the images representing the measurements via adaptive thresholds and tools which allow for manually choosing, moving and deleting ROIs complete the repertoire of the program and allow for a fast and convenient selection of ROIs.