Contents

2.1 Introduction

2.2 PhotoZ and the hardware

2.3 Architecture of PhotoZ

2.4 User interface of PhotoZ

2.5 Digital Signal Processing

2.6 EPSP simulation

2.7 Demonstration of object interactions by one example

2.8 Discussion

# Chapter Two Software Development

## 2.1 Introduction

A flexible, fast, and user-friendly computer program to control instruments and analyze the collected data is important for any imaging experiment. The present voltage-imaging study generated a huge amount of data (usually a few gigabytes for one LTP experiment). When the commercial product does not meet the need of experiments, development of a program is a better choice than asking a company to modify or write a program for you. There are several advantages of developing software in the lab. First, the scientist has total and does not need to compromise with the commercial software. Second, the scientist can modify and expand both the hardware and the software to meet the experimental need without waiting for the commercial software provider to modify it. Third, since the program is designed and developed for the lab, the control of instruments and the analysis of date could be easier and faster. Finally, the lab does not need to pay a lot of money for a modified program. However, development of software requires time and expertise and might distract from the research. Therefore, when a scientist decides to develop a program, one must keep in mind that the development of a program is for the experiment, and not the other way around.

## 2.2 PhotoZ and the hardware

PhotoZ was developed for the photodiode array (PDA) system with the C++ programming language. It controls shutters, stimulators, post-amplifiers, and data acquisition processors (DAP). It records optical signals from the photodiode array (464 channels) and one field potential (FP) signal from an amplifier (expandable to 8 channels) (Fig. 2-1). PhotoZ accomplishes these tasks through communication with DAPs by calling functions in DAPIO32.lib, a commercial library provided with the DAPs (Microstar Laboratories, 1997a).

PhotoZ controls two DAPs in the current version. A DAP 3200e/214 sends TTL (transistor-transistor logic) signals to control shutters, stimulators, and post-amplifiers (Microstar Laboratories, 1994). A DAP 5400a receives optical signals from post-amplifiers and the FP signal (Microstar Laboratories, 2002). In order to control these two DAPs, PhotoZ needs codes written in the language that DAPs can understand. DAPs’ native language is DAPL (Microstar Laboratories, 1997b, c, d). The procedure to communicate with DAPs is first connecting DAPs, then writing DAPL codes to files (\*.dap files in the \PhotoZ folder), transferring the DAPL codes to the DAPs, starting tasks, acquiring data from the DAPs, and finally disconnecting the DAPs.

The best way to understand how the program communicates with the hardware is to modify one variable at a time, trigger an event, follow the function calls, and learn the interactions between them. After explaining the organization of PhotoZ, one example is discussed to demonstrate how PhotoZ controls hardware and acquires data.

## 2.3 Architecture of PhotoZ

C++ is an object-oriented programming (OOP) language. Therefore, PhotoZ first defines classes and then creates objects from classes. Figure 2-2 displays the classes and their relationships in PhotoZ (see also Table 2-1). There is one MainController object, which manages most interactions between objects. In addition to the MainController object, PhotoZ has twelve global objects. Five of them contain local objects and are discussed below.

The ArrayWindow object manages the display in the array window (Fig. 2-3). It contains 472 diode objects. 464 of them display signals from photodiodes and 8 objects display analog inputs, such as field potentials. The ArrayWindow object also contains one Image object, which loads, manipulates, and displays a CCD image as the background in the array window (Fig. 2-3).

The ColorWindow object is in charge of rendering and displaying data features in the map/movie window (Fig. 2-3). It contains 464 GridPoint objects, which are used to construct a three-dimensional map that displays several features of the optical signals, including the spatial distribution of amplitudes, LTP, resting light intensities (RLI), and peak latencies. PhotoZ renders these figures with OpenGL in the map/movie window (Table. 2-2). In the movie mode, PhotoZ constructs the three-dimensional map and renders the amplitude distribution continuously. It can thus play a movie of signal propagation and reveal the dynamics of electrical activity in the slice.

The DapController object controls digital outputs of the DAP 3200e/214. It currently has four DapChannel objects, which control four TTL outputs (one to post-amplifiers (reset), one to the shutter, and two to the stimulators). The DapController object also controls the hardware trigger and the gain control of the post-amplifiers, which occupy two TTL outputs (Fig. 2-1). This leaves two spare TTL outputs, which PhotoZ can use in the future.

The DataArray object stores data in acquired hardware memory and analyzes signal properties of the photodiode array. It manages 472 Data objects, among them 464 Data objects handle optical signals and eight handle other analog inputs. The DataArray object acquires and manages the hardware memory (CPU cache or RAM) from the computer and assigns it to Data objects. The DataArray is in charge of calculation, processing, and comparison of properties returned by Data objects, while Data objects process and measure properties of signals from single channels. For example, one Data object measures the maximal amplitude of one signal from a photodiode, and the DataArray object then compares the maximal amplitudes from all Data objects and saves the maximum of them.

The SignalProcessor object is responsible for digital signal processing of signals. It contains one Filter object and one PolynomialFitting object. The PolynomialFitting object uses a third-degree polynomial function to fit the baseline of optical signals and correct the baseline drift (Fig. 2-4). The Filter object handles temporal filtering. There are two types of temporal filters, which are binomial and moving average filters. The SignalProcessor object also performs spatial filtering, using a two-dimensional Gaussian filter to reduce noise.

The Color object controls colors of traces and the color scale bars in the array window and the map/movie window. It contains four arrays of RGB objects. One RGB object array is in charge of assigning colors to traces. Three others are used to create the color scale bars (color scale 1, color scale 2, and blue to red) for the array window and the map/movie window.

## 2.4 User interface of PhotoZ

The user interface of PhotoZ is created by FLTK (Table 2-2). The interface comprises six main areas (Fig. 2-3) and their properties are listed below. Detail descriptions of how to use the interface are in the user manual.

### Array Window

There are 472 signals in the window with 464 of them from the PDA. Each diode signal is displayed at a specific location. The background of the window can be set according to a CCD image, maximal amplitudes, spike amplitude, resting light intensity, etc.

### Menu Bar

There are three pull-down menus in the menu bar. The File menu can open a data file, create a folder, and exit the program. The Preference menu can save and load a preference file (\*.pre). The Save menu can save the array window or the trace window to an image file.

### Acquisition Panel

The Acquisition Panel contains two tabs. The Acquisition tab controls the procedure of recording. The Analysis tab provides some routines for efficient data analysis.

### File Control Panel

This panel controls the slice number, the location number, the record number, and the trial number, with reference to actual experiments.

### Main Panel

This panel contains five tabs. The Array tab controls the display of the Array Window. The DSP (digital signal processing) tab controls measurement settings, amplitude corrections, temporal filtering, and spatial filtering. The Baseline tab controls the baseline correction of signals. The Image tab controls the size and the position of the CCD image in the Array window. It also controls the input and output image formats. Finally, the Color tab controls the color scale bar in the Array Window.

### Versatile Panel

There are three tabs in this panel. The Trace tab displays traces of selected diodes from the Array Window (selected by simple mouse clicking on diode locations). It also controls the simulation of EPSP components. The DAP tab controls the settings of inputs and outputs of DAPs. The Map & Movie tab displays and controls the map/movie window.

During the development of PhotoZ, the design and coding of the user interface was greatly accelerated by using a small program, FLUID (Fast Light User Interface Designer, http://www.fltk.org). The procedure to modify or expand the user interface first involves modifying or adding controls with FLUID, and then editing callback functions (the functions usually call subroutines of the MainController object), editing or creating a subroutine in the MainController object, and finally updating the user interface (for example, the Array window needs updating after changing the settings of temporal filtering). This procedure is usually tedious. However, with careful design and coding, the effort is paid off when the user spends much less time configuring the software, acquiring data, and analyzing the data.

## 2.5 Digital signal processing

It is essential to extract physiological signals from recorded signals by signal processing when the recorded signals are distorted or the noise is too large. The post-amplifiers are equipped with four-pole Bessel low-pass filters, whose cut-off frequencies are controlled by a TTL input from a pulse generator. This analog signal processing is necessary to prevent aliasing distortion and reduce high-frequency noise when the A/D converters of the DAPs sample the input signals. After analog signal processing, digital signal processing can accomplish the remaining processing tasks. PhotoZ performs three types of digital signal processing. They are baseline correction, temporal filtering and spatial filtering, which are discussed below.

### Baseline correction

Even a tiny drift in an optical signal leads to severe distortion because the voltage-imaging signal is so small (I/I is on the order of 10-3). The circuits in post-amplifiers have an AC coupling feature, which can partially correct the baseline drift. However, the AC coupling also distorts of the physiological signal. To correct the baseline drift, PhotoZ uses a third-degree polynomial function to fit the baseline and correct the drift (Fig. 2-4). PhotoZ excludes the physiological signal when fitting the baseline. Then, the baseline drift is subtracted from the recorded signal. In this way, the baseline drift can be corrected without distortion of the physiological signal.

### Temporal filtering

PhotoZ provides two types of temporal low-pass filters, a moving average filter and a binomial filter. The moving average filter is faster and causes less distortion in the time domain when the sampling rate is high enough. However, its performance is poor in the frequency domain. The equation of the moving average filter follows,

where M is the width of the filter.

The binomial filter is close to a Gaussian filter when the width of the filter is wide. This filter has similar shapes in the time domain and the space domain. Therefore, the binomial filter performs well in both domains. However, the required computation power increases when the width of the filter increases. This is especially true when the sampling rate is high and a wider filter is required. The equation of the binomial filter follows.

;

### Spatial filtering

PhotoZ uses a two-dimensional Gaussian filter to reduce spatial noise. The equation of the filter follows.

;

i indexes the time point. j indexes the six neighboring diodes.  is the standard deviation of the Gaussian function. Diodes at the border of PDA have less than six neighbors so the sum is over fewer terms. In the analysis of optical data, signals from some diodes have a very bad baseline or noise. PhotoZ can ignore these diodes. In this situation, PhotoZ replaces the bad signal with the average of signals from the surrounding neighbors.

## 2.6 EPSP simulation

The optical signals of voltage-imaging reflect EPSPs with or without spikes. In order to assess the magnitudes of these two contributions, PhotoZ can fit the EPSP component with a simple RC circuit model and decompose the optical signals into EPSPs and spikes (Fig. 2-5A). With absorbance voltage-sensitive dyes, the optical signal (I/I) is proportional to the membrane potential change (Vm) within the physiological range of voltages. Therefore, the relation between the optical signal and the membrane potential change can be represented as follows.

 is a constant, which is determined by the dye properties. To simulate the EPSP component of the optical signal, PhotoZ calculates the membrane potential (Vm(t)) by numerical integration of the following differential equation.

In the equation, Cm is the membrane capacitance, gleak is the membrane leakage conductance, Eleak is the reversal potential of the leakage conductance (-65 mV), gsyn is the synaptic conductance, and Esyn is the reversal potential of the synaptic conductance (0 mV). The synaptic conductance was modeled with an -function.

t is time, and Gsyn and  are free parameters. The time course of a simulated EPSP depends on , , Gsyn, and the ratios of Cm to gleak and of Gsyn to gleak. Since  and Gsyn only affect the amplitude of an EPSP, they are incorporated into one variable, amp. Varying Gsyn/gleak in the range of 0.01 ~ 5.0 usually has little impact on the time course of Vm(t), so I fixed it at a constant value of 1.0 (Fig. 2-5 B). The best fit was determined by varying the onset time point, , amp, and Cm/gleak to minimize the sum-of-square error. When calculating the error, PhotoZ ignores the period containing the spike.

Fig. 2-5 B uses an optical signal to illustrate how each variable affects the fitting. The sample trace is best fitted by adopting the parameters shown on the right of Fig. 2-5 A. The onset time determines the point when the EPSP starts. The amplitude controls the size of a simulated EPSP. The time constant of the -function () mainly affects the rising rate of the EPSP. The ratio of Cm to gleak predominantly controls the falling rate of the simulated EPSP. Finally, the last panel of Fig. 2-5 B demonstrates that the ratio of gsyn to gleak only weakly affects the time course of the simulated EPSP in a certain range.

## 2.7 Demonstration of object interactions by one example

The recording task is discussed below to demonstrate how objects in PhotoZ communicate with each other and how PhotoZ communicates with DAPs. When you click the “Record” button, it triggers an event and PhotoZ tells the DAPs to send TTL signals and record signals (Fig. 2-6).

### The graphic user interface (GUI)

First, use FLUID to open the GUI file, UserInterface.fl. Double-click the mainWindow if the main window is not shown. By double-clicking the “Record” button, a Fl\_Button window appears. Click the C++ tab and you can see the callback function window. The callback function of the “Record” button calls the record function of the MainController object, mc (mc->record()). That is, when the “Record” button is clicked, an event is triggered, which starts the callback function of the “Record” button. Then, the callback function asks mc to perform record().

### The function record() in MainController

The MainController object handles most communication between objects, so the GUI calls a function of the object. record() of mc is declared in the header file, MainController.h, and defined in MainControllerAcqui.cpp. Inside record(), PhotoZ first checks whether the “Schedule” box is selected. If it is not selected, PhotoZ calls acquiOneRecord() and acquires one record. If the box is selected, record() gets the interval between records by calling recControl->getIntRecords() and gets the maximum number of records by calling recControl->getNumRecords(). record() then saves the time and starts the first recording by calling acquiOneRecord().

After completing one record, PhotoZ checks whether the “RLI” box is selected. If it is selected, it takes a resting-light intensity by calling takeRli(). Then, record() saves the data to a file by calling saveData2File().

After saving the data, record() waits until the time for the next recording comes. During the waiting, record() keeps checking whether the “Schedule” box is unselected or the “Stop” button is clicked. If either one of these two situations arises, PhotoZ stops recording. If none of these two situations arise, PhotoZ increases the number of records by calling increaseNo(RECORD) and keeps recording data until the maximum number of records has been reached. Among the functions inside record(), acquiOneRecord()is further discussed below to demonstrate how PhotoZ controls DAPs.

### The function acquiOneRecord() in MainController

In the acquiOneRecord()function, PhotoZ first gets the information of the number of trials and the interval between trials from the recControl object, which are required for recording control. It then gets the number of points from the dapControl object. Since the raw data from the DAPs is sequential in time (the first data point is from diode 1, the second data point is from diode 2, ...., the 472nd data point is from analog input 8, the 473rd data point is from diode 1, the 474th data point is from diode 2, etc.), PhotoZ acquires a block of memory buffer to store the raw data from the DAPs so it can rearrange the data and save it later. Next, acquiOneRecord()enters a for loop to acquire each trial data. Inside the loop, PhotoZ keeps track of time and checks whether the “Stop” button is clicked.

During each trial, PhotoZ connects and resets the DAPs by calling setDAPs() and resetDAPs() of the object dapControl. It also creates two dap files required to communicate with DAPs by calling dapControl->createAcquiDapFile(), which creates “Record-3200 v5.dap" and "Record-5400 v5.dap” in the \PhotoZ folder. PhotoZ then sends the two files to the two DAPs by calling dapControl->sendFile2Dap(). After sending the dap files (written in DAPL), it is ready to start the data acquisition by calling dapControl->acqui().

The acqui() function takes a while to complete since it covers the whole data acquisition period and needs to wait until the data transfer from the DAP5400a to the computer is completed. After that, PhotoZ rearranges and saves the data in the DataArray object by calling dataArray->arrangeData(). In order to process and display the recorded data, PhotoZ needs to load the recorded data to a block of working memory in DataArray object and process the data by calling dataArray->loadTrialData() and dataArray->process(). Inside the process() function, PhotoZ corrects the baseline drift and performs temporal and spatial filtering. At the end of each trial, PhotoZ updates the GUI and redraw the contents so that it can display the data after each trial.

After all trials are completed, PhotoZ releases the buffer memory, which it acquires before the data acquisition. Without releasing the buffer memory, PhotoZ would keep acquiring memory from the system and the computer could crash. It is useful to show the average signal after completing data acquisition. Therefore, PhotoZ loads the averaged data into the working memory of the DataArray object and processes it. Finally, PhotoZ updates the GUI and redraw the content.

## 2.8 Discussion

It is a challenge to write a program from scratch that controls hardware and performs most data analysis. PhotoZ has undergone four major revisions since the first version. After each major revision, there are several minor revisions. The current version of PhotoZ is 5.20. To control DAPs, the software designer needs to learn the language that the DAPs understand (DAPL) and to learn how to communicate with DAPs (DAPIO32.lib) (Table 2-2). To create the graphic user interface (GUI), the designer needs to search for a proper external library and learn to use it (FLTK). Although it is possible to create the GUI by using internal libraries of Windows, it is extremely tedious and inefficient. To load and manipulate CCD images and save rendered images in different image formats, external libraries (paintlib.lib) are required, because C++ does not support image input/output. Finally, PhotoZ uses OpenGL to render the images and movies in the map/movie window (opengl32.lib, fltkgl.lib).

PhotoZ is a program providing an integrated environment for voltage-imaging experiments with a photodiode array system. It controls instruments, records signals, analyzes the signals, and displays them. In addition, PhotoZ also strongly supports LTP experiments with one or two stimulating electrodes. Because PhotoZ is very flexible, the combination of PhotoZ and the PDA provides flexibility in conducting LTP experiments with voltage-imaging technique.

Figure 2-1. Hardware connection diagram



PhotoZ controls hardware and acquires data through calling functions in DAPIO32.lib. There are two DAP boards in the current configuration. A DAP 5400a acquires data from 8 analog inputs directly (including one signal from a patch-clamp amplifier) and 464 analog inputs from post-amplifiers, which receive signals from the photodiode array (PDA). A DAP 3200e/214 sends TTL (transistor-transistor logic) outputs to shutter controllers, pulse generators (controlling stimulators), resets the post-amplifiers, and controls the gains of the post-amplifiers. The DAP 3200e/214 also sends a hardware trigger signal to the DAP 5400a to initiate the acquisition of data.

Figure 2-2. Architecture of PhotoZ



Figure 2-2. Architecture of PhotoZ

PhotoZ has 20 classes. Thirteen of them are global objects. Although the MainController object contains no local objects, it is at the first level because it manages other objects and handles the communication between them. Five of the global objects have local objects. Therefore, the architecture of PhotoZ can be viewed as a three-level organization.

Figure 2-3. Snapshot of PhotoZ



The graphic user interface can be divided into six main areas, the Array Window, the Menu Bar, the Acquisition Panel, the File Control Panel, the Main Panel, and the Versatile Panel. Each main area can have several tabs or other control widgets. The function of each control widget is in the user manual.

Figure 2-4. Baseline correction with a third-degree polynomial function



Ignored window

The black trace is the original recorded optical signal. Although the baseline drift is tiny (less than 1%), the signal is severely distorted (the physiological signal is less than 0.1%). PhotoZ ignores the period containing the physiological signal and fits the baseline with a third-degree polynomial function (green trace). The baseline drift is subtracted from the original optical signal, producing the red trace as the corrected signal.

Figure 2-5. Simulation of EPSP



Figure 2-5. Simulation of EPSP

A. (Left) There are five elements in the model circuit simulating the EPSP component. Cm is the membrane capacitance. Gleak is the leaking conductance and Eleak is the reversal potential (-65 mV). Gsyn is the synaptic conductance and Esyn is the reversal potential. The value of Gsyn is time-dependent and calculated according to an -function. Vm is the membrane potential. (Right) These are values of the parameters that determine the best fit to the EPSP component of the optical signal in B.

B. The time course of the simulated EPSP is determined by five parameters. The onset time controls the latency. The amplitude determines the size. The time constant () mainly determines the rising rate and the ratio of Cm to gleak affects the falling rate. Finally, the ratio of gsyn to gleak only weakly influences the time course of the simulated EPSP in a certain range. During fitting, PhotoZ usually fixes the ratio of gsyn to gleak and searches a four-dimensional parameter space repetitively to find a set of parameters those results in the minimal sum-of-square error between the simulated EPSP and the optical signal.

Figure 2-6. Flowchart of “Record”



Figure 2-6. Flowchart of “Record”

After clicking “Record”, the GUI generates an event and calls the record() function in the MainController object. The function checks the “Schedule” box and determines whether to repeat the recording or not. If repetition is not required, it acquires one record and ends. If the “Schedule” box is selected, the function starts counting time and acquiring data. After finishing one record, the record() function checks whether the “RLI” box is selected. If it is selected, the function takes the RLI and saves the data to a file. After saving the data, the function increments the record number and checks whether the record number exceeds the maximum number of records. If it does not exceed the maximum number, the function keeps counting time and repeating data acquisition. If the record number exceeds the maximum, the function quits the loop and returns the control to the GUI.

Figure 2-7. Flowchart of acquiOneRecord()



Figure 2-7. Flowchart of acquiOneRecord()

The acquiOneRecord() function in the MainController object directly communicates with three objects. The RecControl object controls the procedure of recording and contains the information of the number of trials and the interval between trials. Thus, the acquiOneRecord() function asks the RecControl object for these information. The DapControl object is responsible for the communication with the DAPs. The acquiOneRecord() function asks it for the number of points in each trial in order to acquire a block of temporary memory to store the raw data from the DAP 5400a. The function then connects, resets, and sends codes in DAPL to the DAPs through the DapControl object. After these initiation steps, the acquiOneRecord() function acquires raw data from the DAP 5400a, also through the DapControl object. Then, the function sends the raw data to the DataArray object for rearrangement and storage. The acquiOneRecord() function also asks the DataArray object to load the data to the working memory and process it. After that, the function plots the data, updates the GUI (not shown), and releases the DAPs. The function keeps acquiring data until it completes all trials. Finally, the acquiOneRecord() function releases the temporary memory, updates the GUI, and displays the average signals.

Table 2-1. Classes and their functions

| Class name | Functions |
| --- | --- |
| MainController | The main controller object handles most events and communications between objects. |
| UserInterface | The graphical user interface object handles inputs from the user and organizes the interface. |
| ArrayWindow | The array window object contains 472 diode objects and displays the data in the array window. |
| Diode | The diode object handles events of one diode. It also displays the trace and several properties as background. |
| Image | The image object controls and displays the CCD image in the array window. |
| TraceWindow | The trace window object displays traces, slopes, simulated EPSPs, and time course of several properties. |
| ColorWindow | The color window object processes and displays rendered figures of several properties. It can also play movies showing the dynamic change of optical signals. |
| DapController | The DAP controller object controls DAPs, which get inputs from the photodiode array and sends TTL outputs to several devices (shutters, stimulators, and etc). |
| DapChannel | This object controls one TTL output channel. |
| DataArray | The data array object controls and manages 472 data objects. |
| Data | The data object contains the raw and processed data from one channel. It also analyzes the data and stores the properties. |
| FileController | This object controls the procedures of saving and loading data. |
| SignalProcessor | It controls the digital signal processing of the signal. |
| Filter | It performs temporal low-pass filtering of signals. |
| PolynomialFitting | This object uses three-degree polynomial functions to fit the baseline of signals and corrects the signal drift. |
| RecControl | This object controls the procedure of recording. |
| WindowExporter | This object saves the displayed data in image files (BMP, TIFF, PNG and JPEG). |
| Color | The color object controls and adjusts the colors of traces and color scale bars. |
| GridPoint | This object is used to construct a 3-D map, which is required to create the color map and movie in the color window object. |
| RGB | This object is used to set the colors of traces, diodes and color scale bars. |

Table 2-2. External libraries

|  |  |  |
| --- | --- | --- |
| Library name | Purpose | Websites |
| dapio32.lib | Used to control DAPs. | http://www.mstarlabs.com |
| opengl32.lib | Required to render figures in the map/movie window. | http://www.opengl.org |
| fltk.lib | Used to construct the graphical user interface. | http://www.fltk.org |
| fltkgl.lib | Required to use OpenGL in FLTK. | http://www.fltk.org |
| paintlib.lib | Image input and output. | http://www.paintlib.de |
| DAPIO32.dll | This dynamic link library is required to communicate with DAPs. | http://www.mstarlabs.com |