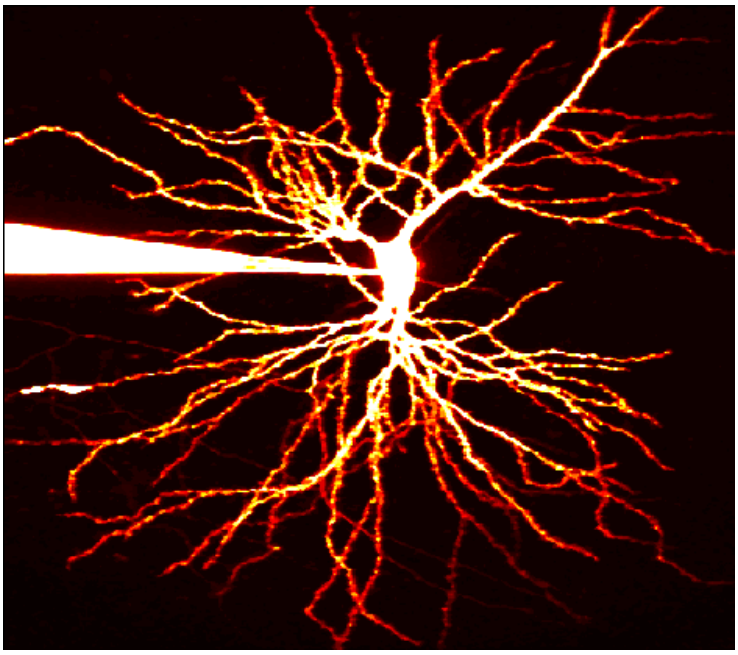
ScanImage



Version 3.0

Release 1.0

Introduction

ScanImage controls a laser scanning microscope such as a 2-photon or confocal. It is written entirely in Matlab 6.1 and makes use of standard National Instruments boards for data acquisition and scanning output. The software generates the voltage used to drive the scan mirrors, acquires the raw data from the photomultiplier tubes, and processes these signals to produce sub-micron resolution images. Up to three channels (12-bits each) can be acquired simultaneously, and the software is written to be easily expandable to the maximum number of channels the NIDAQ board supports. The software runs at nearly real-time rates even when acquiring on three channels simultaneously, updating the images with minimal delay from the time they are actually obtained. Virtually no specialized hardware is required for either scan mirror motion or data acquisition.

This software has been published in the journal Biomedical Engineering Online and the reference that needs to be cited when using *ScanImage* is:

Pologruto, T.A., Sabatini, B.L., & Svoboda, K (2003) *ScanImage*: Flexible software for operating laser scanning microscopes. Biomedical Engineering Online **2**:13.

Contents

1. Software and Hardware Requirements

- Since *ScanImage* requires heavy computation from the CPU, a computer equipped with a **Pentium III** (800MHz) **Processor** or better, running **Windows 2000,** is needed.

- Data acquisition and output is achieved using a **6110E** data acquisition board from **National Instruments**.
    - The **BNC-2090 Breakout Box** is preferred for attaching the actual BNC cables to the board and the scan mirrors & PMTs.

Sutter MP-285 stepper motor controller is used by *ScanImage* to implement XYZ motion control via the serial port.

- **Matlab** (v6.1 or better)
    - Need the supplemental **Data Acquisition Toolbox (**v2.0 or better**).**
        - **The Image Processing Toolbox** (current version) is not necessary for data acquisition, but would be very useful for image manipulations and user function writing.
        - Also suggested are the **Statistics** and **Curve Fitting Toolboxes** (current versions).

- **National Instruments NIDAQ** software (v6.7 or later)
    - Matlab communicates to the boards via their NI drivers, included with this software.

- **Stanford Instruments SR570 Amplifiers**
    - Used for amplification of the current from the PMTs that is fed into the NI boards.

- Microscope, optics, detection devices, etc… also need to be installed and functional.

2. Setting Up the Program

2.1 Install the Data Acquisition Boards and NIDAQ Software

Install the NI **6110E** data acquisition board (5 MHz input rate) into a free PCI slot on the computer. Load the NIDAQ v6.7 or later driver software and look at the "Measurements and Automation" (MAX) program to see the location of the board you just installed. Once you open the program, click the "+" sign next to "Devices and Interfaces" to list all installed NI boards. Note the index of the board you will use to acquire data and output the scanning mirror waveforms. This board index is what Matlab will use to configure the NI board.

If you are installing two boards, make a note of the indices of each board to be used. *ScanImage* needs only one board (the **6110E**) to function, so most users can disregard this section and can proceed to **Section 2.2**. The second board can be a **PCI-MIO-16-E** and is used with *ScanImage* for interaction with scanning hardware and other additional controls. The name of each board is also listed when you click on the "+" from the MAX interface. If the boards have the same name, please note and label the boards as such to avoid confusion in wiring and configuration. As an alternative, an analog output (**PCI 6713**) board may be installed, as this provides more flexibility in integrating new time-dependent devices.

2.2 Install Matlab 6.1 or later

Matlab (6.1 or later) should then be installed with the appropriate toolboxes (Image Processing and Data Acquisition Toolboxes are required). On Windows 2000, an administrator needs to install and *start* Matlab before other users run it. If the administrator installs but does not start Matlab, other users get the following error message about writing to the registry when they start Matlab (they can run Matlab but must first dismiss the error message).

```
Unable to set Registry value
CLSID\{B3044B60-97FF-11D1-8146-00600815A7AD}\LocalServer32.
You may not have sufficient privileges. Rerun MATLAB as a user
with Admin access.
```

In addition, the administrator needs to run the command `daqhwinfo('nidaq')` in order for subsequent users to be able to do so (see below for usage details). Installation of all software should be done from an administrator login on Windows 2000 systems.

Once Matlab is installed, you should check that Matlab recognizes the NI boards you installed. Matlab should automatically find installed data acquisition (DAQ) devices on the computer; type `daqhwinfo('nidaq')` from the Matlab command line to check if Matlab has detected the installed data acquisition board(s). If done successfully, they will appear in the `'BoardNames'` field. If they do not, try restarting the computer and checking again. This needs to be done on Windows 2000 machines while logged in as the administrator.

NOTE: *For stability, Matlab 6.5 requires updated versions of the rxtxSerial.dll and comm.jar, obtained from Mathworks. Not having these patches will likely result in crashing when communicating with the motor.*

2.3 Loading and Configuring *ScanImage*

The file structure for *ScanImage* is contained in a directory called **ScanImage** that needs to be placed in a Matlab path. If you downloaded *ScanImage* from the web as a zip file, you need to unzip (extract) it to the Matlab work directory that will automatically create the **ScanImage** directory. It is highly recommended that it be placed in the Matlab **work** directory, although this is not necessary. Once the files are placed correctly, Matlab can be started and the path is set by going to the **File** menu in the Matlab command window and selecting the submenu **Set Path**. Press the **Add with Subfolders** button and highlight **ScanImage** directory to add all the files within *ScanImage* to the Matlab path.

As of **ScanImage** 3 there are separate modules upon which the program depends. These include the class @daqmanager and the library. As with all Matlab classes, daqmanager must be added to a folder on the Matlab path, but should not be added to the path itself. Typically, it is placed into a UserClasses folder under Matlab, which is added to the path. The library folder, and all its subfolders, must be added to the path.

Next open the initialization file and configure it for the computer you are using by typing `edit standard.ini` from the command line. This brings up an editable window with the text for `standard.ini` displayed. A few of the fields need to be checked before launching *ScanImage* to ensure proper function. For convenience, all the fields that need to be updated are located at the top of the file. Finding the fields is easily done by pressing **Ctrl+F** and typing the fieldname into the find box, although reading through the file will be helpful later.

The following is an excerpt of the text from the standard.ini file with the fields that need to be set and their definitions.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The following variables are the ones that typically need to
%% be changed for a new installation.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

structure init
 % acquisitionBoardIndex, mirrorOutputBoardIndex, &
 % triggerBoardIndex are most likely the same and are equal to the
 % index of the  fast (> 5 MHz) Multifunction Board (e.g. NI PCI 6110)
 % determined by calling daqhwinfo('nidaq') from the MATLAB command line.
 acquisitionBoardIndex=2  % Index to DAQ board for use with PMT acquisition.
 mirrorOutputBoardIndex=2 % Index to DAQ board for use with galvo mirror output.
 triggerBoardIndex=2      % Index to DAQ board for use as TTL trigger.

 % Pockels Cell (EOM) Configurations
 pockelsOn=1 % Flag to determine whether Pockels Cell is being used: 1 = yes, 0 = no

 % pockelsBoardIndex is most likely equal to the index of the slow (100-250 kHz)
 % Multifunction Board (e.g. NI PCI MIO 16E-4 or 6014) determined by calling
 % daqhwinfo('nidaq') from the command line.

 % Modulator Specifics. Each of N EOMs (Pockels Cells) must have pockelsBoardIndexN,
 % pockelsChannelIndexN, photodiodeInputChannelN, photodiodeInputBoardIdN,
powerConversionN,
```

```
 % rejected_lightN, maxPowerN, maxLimitN, and photodiodeOffsetN included below.

 structure eom
  numberOfBeams=1           % Sets how many Pockels cells are on the rig.
  scanLaserBeam=1           % Beam used for imaging.
  pockelsBoardIndex1=1      % Output DAQ board for use with scan laser Pockels cell.
  pockelsChannelIndex1=3    % Output DAQ channel for use to drive scan laser Pockels
cell.
  photodiodeInputBoardId1=2 % Input DAQ board where photocurrent is input.
  photodiodeInputChannel1=3 % Channel on Input DAQ board where photocurrent is input.
  powerConversion1=450      % Conversion factor:
                            %  <powerConversion> * <photodiodeVoltage> =
<powerInMilliWatts>.
  rejected_light1=0         % The calibration input comes from accepted/rejected
light.
                            % 0 for accepted, 1 for rejected
  maxPower1=50              % The startup values can be set here...
  maxLimit1=100             % Same as above.
  photodiodeOffset1=0       % The 'rest' voltage from the photodiode for beam 1
                            % (this covers amplifier offset, ambient light, etc).
 endstructure

endstructure

structure motor
 % Stage and Z-Focus controls using a Sutter MP285.
 % N.B. motorOn must be set to 1 if Z-stacks are to be taken.
 motorOn=1   % Flag to determine whether MP285 is being used 1 = yes, 0=no
 port='COM1' % Serial port for MP285 communication
endstructure

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% End of section commonly edited on new installation %%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

`maximumNumberOfInputChannels`

This should be set to the number of inputs you will use. Typically 3 are used: one for Green, one for Red, and one for trans-DIC images. 3 is the maximum for this generation DAQ board (6110E).

`autoReadPMTOffsets`

This is set to 1 if wish to read the dark (shutter closed) PMT settings before acquisition, and to 0 if not.

`acquisitionBoardIndex`
`mirrorOutputBoardIndex`
`triggerBoardIndex`

These need to be set to the board indices you recorded from the MAX software. The `acquisitionBoardIndex`, `mirrorOutputBoardIndex`, and `triggerBoardIndex` should be the same index (5 MHz board like 6110E), and ANY others should be set to the slow board index (like the PCI-MIO-16-E board).

`numberOfBeams`

This determines the number of laser beam lines under the control of *ScanImage*. As of this release, only 1 is supported. Future releases may support more.

`ScanLaserBeam`

This determines which, of all available beams, is the scan laser.

`pockelsBoardIndex1`

The NIDAQ board used for Pockels cell control output.

`pockelsChannelIndex1`

The output channel for Pockels cell control data.

`photodiodeInputBoardId1`

This is the board to which input from a photodiode (or other appropriate power monitoring device) is attached. It is mainly used for calibration of the Pockels cell (usually done on startup, unless configured otherwise).

`photodiodeInputChannel1`

This is the channel to which the photodiode (discussed above) is attached.

`powerConversion1`

This is a factor used to convert between an input voltage from the photodiode to a laser power (in milliwatts). The factor is used such that powerConversion1 multiplied by the input voltage results in the power in mW.

`rejected_light1`

This determines placement of the photodiode. A Pockels cell has two output ports, one of which continues along the normal laser path (this is the accepted light), the other is roughly 45° off from the normal laser path (this is the rejected light). The photodiode may collect light from either of these ports.

`maxPower1`

This is the default setting for the maximum power, in percent of maximum power. Typically the maximum power is set using the Power Control GUI, but a default setting may be configured here.

`maxLimit1`

This is the default setting for the "soft limit" on the maximum power, in percent of maximum power. Typically, this variable is set using the Power Control GUI, but a default setting may be configured here.

`photodiodeOffset1`

This variable compensates for any offset in the transimpedence amplifier associated with the photodiode, as well as any offset due to ambient light. To determine this value, the voltage coming out of the amplifier should be measured with the laser off (or blocked before reaching the photodiode). It is recommended to make the measurement under the rooms typical lighting conditions, to account for ambient light.

The remaining fields control the use of the **Sutter MP-285 Stepper Motor** (`motorOn`, `port`, and `baud`). Set these as follows:

1) If you are using the **MP-285**, set `motorOn=1`. If not, set `motorOn=0`.
    a. Determine to which serial port the **MP-285** is connected and set `port='COMX'`, where `X` is the port number.
    b. Set `baud=19200.`
    c. Set the **Baud Rate** on the **MP-285** Controller to 19200.
        i. This is done by:
            1. Selecting PRGM on the front of the MP-285.
            2. Hitting 0 2 times to highlight Setup and pressing ENTR.
            3. Hitting 0 3 times to select Utilities and pressing ENTR.
            4. Pressing ENTR to select Baud.
            5. Pressing 0 4 times to select 192000 and pressing ENTR.
            6. Pressing ESC 4 times to exit the configuration.
        ii. This must be done, since *ScanImage* requires this baud rate for efficient communication.

N.B. *ScanImage* is intended to be used with a **MP-285** motorized stage, and setting this field to off will limit the abilities of *ScanImage* (i.e. inability to collect z stacks or use cycles).

If you are using a motor, be sure the devices are turned on prior to starting the software.
The **MP-285** needs to be turned on but the green "Move" button should not be pressed until after the software is loaded (The display on the **MP-285** reads `*** MP-285 Controller ***` in this state). The software will update the **MP-285** screen with coordinates if communication is established correctly, but manual control will not work until the green 'Move' button is pressed on the front of the **MP-285**. This should be done after *ScanImage* is started.

2.4 Starting *ScanImage*

We have included a few functions in the **ScanImage** directory that are used for starting the program. Depending on your preference, a graphical interface can appear that allows you to select which application to run within Matlab, or you may open the software from the command line.

If you wish to use the Opening GUI interface, follow the following steps (this is recommended if Matlab was installed primarily to run *ScanImage*):

1) There is a top-level function called **startup.m** that will be executed whenever Matlab is opened. It can be used to set general Matlab preferences or to open other applications.

2) Open the startup file by typing `edit startup` on the command line.

3) Add the following lines of code to this file and **Save** the changes.

```
global gh
gh.openingGUI = guihandles(openingGUI);
```

4) Close and restart Matlab.

When Matlab reboots, a GUI should appear allowing you to select the application to run. Click on the *ScanImage* software to start the program.

If you do not wish to have the Opening GUI, then the *ScanImage* software may be started by simply typing `ScanImage` at the command line to start the program.

Regardless of the method used to start *ScanImage,* the software will open a window asking for you to 'Choose User File', which is a file with extension .usr. This file contains information about the layout of *ScanImage* and is discussed in detail in **Sections 3** and **6**. If you do not select one (since you have not made one yet), it uses default settings. Hitting 'Cancel' will start the *ScanImage* set-up routine. The progress is reported graphically.

Start-up proceeds as follows:

1) User selection of .usr file.
2) GUI setup.
3) Reading the .ini file and initializing the GUIs.
4) Configuring the **Sutter MP-285**.
5) Setting up the Data Acquisition and Scanning devices (lines 52-60).
6) Reading User Settings.
7) Initializing the MP-285.
8) Finishing-up and setting clocks.

To see the start code, you can type `edit ScanImage` at the command line.

Once **Main Controls** say "Ready to Use" and the progress bar disappears the software is loaded correctly.

2.5 Wiring *ScanImage* Hardware

The final things to configure are the actual input and output channels the software will be using. For the input side, on the **BNC-2090 Breakout Box**, there are 16 input channels, labeled ACH0-15. Attach the red filtered PMT BNC cable to the ACH0 (Green Channel), the green filtered PMT BNC cable to the ACH1 (Red Channel), and the DIC BNC cable to the ACH2. These channels correspond in the software to Channels 1,2, and 3 respectively, as the board and software have a one-channel offset.

There are only 2 output channels on the **BNC-2090 Breakout Box** labeled DAC0OUT and DAC1OUT, and these are attached to the mirror driving motors. By default, the DAC0OUT is for the fast scanning (X) mirror, while the DAC1OUT is for the slow scanning (Y) mirror, although this can be changed in the software (see the **Basic Configuration GUI** in **Section 4** for details).

The system uses an internal digital (TTL) trigger for timed output and acquisition. This comes from the digital inputs/outputs on the **BNC-2090 Breakout Box** that are located on the right hand side. The connections should be made as follows:

1) Connect the DIO0 output (TTL output) to the PFI0/TRIG1 BNC using the appropriate adaptors.
    a. This is the trigger for the output of the scan mirror data.
2) Connect the DIO0 output to the PFI6 wire hole on the digital input/output section of the board.
    a. This is the trigger for the input from the PMTs.
3) Connect DGRN1 to the ground.

N.B. The TTL digital hardware trigger is a *falling-edge trigger*, meaning that a high-to-low transition is necessary to fire the trigger and begin acquisition and output.

The shutters are also controlled with the acquisition board through the digital outputs via a TTL pulse. DIO1 is the default channel to use for the epi-fluorescence shutter and DIO2 is the default for the laser scanning shutter, although these can be set in the `standard.ini` file (see below).

The shutter controllers are usually connected by converting the wire outputs from the **BNC 2090** breakout box to BNC cables, and then connecting them to the proper shutter driver devices.

The gating of the shutters can be set manually by opening standard.ini (type `edit standard.ini` at the command line) and finding the **shutter** structure using **Ctrl + F**. You should see the following:

```
structure shutter
     open=1
     closed=0
     shutterLineIndex=1
     shutterLine
     epiShutterOpen=1
     epiShutterClosed=0
     epiShutterLineIndex=2
     epiShutterLine
endstructure
```

The open and closed (and **epiShutterOpen** and **epiShutterClosed**) fields set what is defined as open and closed digitally. If your controller is open with a high signal (i.e. 1), that is the default used here. If not, switch the ones and zeros to match your shutter controller logic.

2.6 External Triggering

If the user wishes to trigger *ScanImage* from another application or device using their own TTL trigger, that can be accomplished very simply. The wire coming from DIO0 which carries the TTL trigger just needs to be removed and connected to any TTL Trigger. When you tell Matlab to acquire, it will wait for the pulse to be received from the device to begin acquisition. Note that doing this will also make the focus mode subject to the same trigger. Typically, *ScanImage* is used to trigger other devices using the TTL pulse from channel DIO0.

3. General Logic of *ScanImage*

ScanImage was written to integrate image acquisition and scan design into an environment amenable to sophisticated data analysis, such as Matlab. *ScanImage* was written to allow the user a large amount of flexibility in designing imaging experiments. It includes unique features that make it ideal for the most demanding experiments.

Two things dictate the flow of the program: user input or interaction with the GUIs and the programs own timing of responses to those inputs. Every time the user updates a field in a GUI, whether it is inputting a value into a field, hitting a button, or checking a box, a callback function is executed. Many of these callbacks do not do anything the user can see, but rather compute various parameters the software needs to accomplish a successful acquisition. Fields that are editable follow the standard Windows convention of being colored white, while un-editable fields (buttons, boxes, etc…) are either grayed out or hidden. This latter task is done by the software automatically to maintain stability and control the flow of the program.

Some callbacks do cause noticeable changes in the program. This includes the aforementioned graying out of fields, hiding controls, and opening and closing GUI windows. *ScanImage* may control the program flow utilizing "Modal" GUI windows. When these windows are opened (the **Channel** and **Configuration** windows, for example), no other functions are accessible until the window is closed. In addition, the **Status String** in **Main Controls** (see **Section 4** for details) will display error messages and the progress of the acquisition.

The program uses a number of text files (.ini, .usr, and .cfg) and Matlab .mat files (both described below) to set various acquisition, output, and analysis parameters. The GUIs receive input from the text files for initialization, configuration, and user defaults, and from the .mat files for cycle definitions. The main text file is the .ini file, and it sets what fields will be used in the creation of subsequent text files (like .usr and .cfg files). Which fields from the .ini get put into the other text files is set by the Config flag in the .ini definition.

The display of the GUIs is controlled via the **Settings** menu in **Main Controls**. Selecting the GUI from **Settings** opens it, while using the Windows default close command (mouse click on X) suffices to close the GUI.

3.1 Files

Initialization File (.ini)

This contains the .ini files. These files define the entire state variable with which the program interacts. These files are computer specific and are not meant to be updated regularly. It contains information regarding the tools used on a particular microscope (e.g. motor COM port).

.ini files are text files and are parsed by functions contained in the file_io\initialization directory to enable communication between Matlab functions and the GUIs. The logic is like other schemes in Matlab: there are parameter-value pairs that are parsed by the software. For instance, if you look at the standard.ini file, you will see a tree like (tabbed-in) structure and after a variable definition (like **numberOfFrames**) there is an "=" and then a value. This is the initial value of that variable

and hence its corresponding field in one or more GUIs. Following the variable definition and initial value are a series of other strings (e.g. GUI, Numeric, Callback, …). These are parameters that tell the software to tie the given variable to the value that follows the parameter definition. For example, the parameter GUI is followed by the handle to a particular field in a GUI. When parsed, it will automatically update the state variable whenever the field in the GUI is changed.

The Callback parameter is followed by a function in the Matlab path. Whenever the corresponding variable is updated in a GUI, the function defined here will be executed. This provides a way for users to interface with the software without interfering with its core functions.

Configuration File (.cfg)

The configuration file is a text file like the .ini file, and is actually a subset of that file. It contains fields particular to a given experimental configuration, like the image size, averaging, and the number of channels to acquire. Configurations can be loaded and saved from the **File** menu in the **Main Controls**. Configurations can also be loaded as part of a cycle by choosing **Select** from the **Cycle Controls** window.

User Files (.usr)

The user file is a text file like the .ini file, and is actually a subset of that file. It contains fields particular to a given user's preferred program display, including the positions and sizes of the GUIs, acquisition windows, a favorite configuration, and also if they prefer to start-up in **Standard** or **Cycle** mode. The .usr file can be loaded and saved from the **File** menu in the **Main Controls**. These files are typically loaded upon starting the program.

Cycle File (.cyc)

A cycle is a series of positions that coordinate defined acquisitions with pre-determined positions. The entire process can be timed and repeated to tailor the experiment to the needed situation. The cycle file is a Matlab (.mat) file that contains arrays of values for the parameters that need to be updated and changed when a cycle is executed. The .cyc file contains ordered arrays of parameter values for a particular cycle, while the physical positions to perform each step in the cycle can be loaded from a .pos file or input directly using the **Cycle Controls** window.

When the **Cycle Controls** window is opened, a cycle may be defined, saved, or a new one loaded by going to the **File** menu in **Main Controls**.

Cycles are gone over in depth in **Section 5.**

Position Files (.pos)

From the **Motor Controls**, you can define a list of positions and save those positions in a .mat file. This is used in conjunction with a cycle for design of complicated acquisitions. The current position is written in the file header when the image is saved to disk.

The data is acquired and stored in memory until the next acquisition is begun. The data is logged to disk anytime the stage is moved and/or after all the data is acquired, depending on if a stack of images is taken or not.

3.2 Data Storage

In the **Main Controls** window under **Settings** there are numerous saving options, and checking them will define the defaults for the particular acquisition.

Autosave

This should always be checked unless doing some debugging. This will save the images as defined by the save path and basename. If this is unchecked, users may save the last acquisition by selecting **Save Current Acquisition As**… from the **File** menu in **Main Controls**. Note: If the "Keep all slices in memory" flag is unchecked, the only data to be saved manually will be the last image acquired, *not* the entire acquisition.

Automatic Overwrite

If not checked, a warning will appear if a file is about to be overwritten. Some users may find this annoying, while others a necessity.

Keep all slices in memory

If checked, will store all images of a stack in dynamic memory (RAM), which will limit some big stacks from being acquired properly. If unchecked, the number of slices is not limiting as only a single frame gets stored in memory. The reason for selecting this option is to avoid the overhead of writing to disk, during acquisition.

The acquired image data is stored as 16-bit TIF images. The save path is selected by the user before acquisition, and the saving is done automatically, using the user defined **Basename** and the automatically updating acquisition # (See **Main Controls** for details).

3.3 Header Structure

The relevant information about an acquisition is stored with the data in the form of an image header. The header is a string (`char` in Matlab) that contains all the relevant acquisition parameters in a parameter-value format, using carriage returns as delimiters. The first element (the parameter) is the name of the state variable (like `state.acq.numberOfFrames`) followed by an = sign, the value, and then the carriage return delimiter.

The functions that read the header (or parts of it) from TIF files are included in the software, the most useful of which is `getHeaderVal`, which takes the parameter name as a character (e.g. `'state.acq.numberOfFrames')` and outputs the value.

3.4 Multi-dimensional Image Storage

If you are not using Matlab, the header string may be accessed from the generic tiff header. The field of the TIF header called *Image Description* stores the string header produced by *ScanImage* as a string with a variable length.

Since stacks of images may be acquired on up to three channels simultaneously, *ScanImage* implements an interleaving protocol to save the images from different channels in the same TIF file. The program starts from the first frame on the first channel acquired and loops through the subsequent channels before moving to the second image. It repeats this way until all the data is loaded from memory to disk.

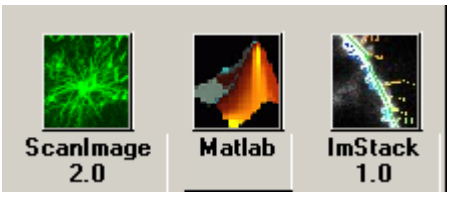
For example, say you acquired a stack of 10 images with 2 frames acquired per stack on 3 channels (60 images total). All 60 of these images are stored in the same TIF file. The order would be the first image of the first channel of the first slice, followed by the first image of the second channel of the first slice, and lastly the first image of the third channel of the first slice. The process is repeated for the second image in the slice, and then for subsequent slices.

If averaging is on, all the frames in a given slice are averaged into one image, and only this image is logged to disk. In the previous example, if averaging would have been selected, there would be a total of 30 images acquired, one frame in each slice for 10 slices on 3 channels.

If max is selected, the max projection is stored for each channel being acquired. By definition, there is only one image per channel for max projections, and the data is interleaved as described above. Max projections are only valid if you acquire 1 frame per slice or if you Kalman average over all the frames in a slice, so if Max is selected but these criteria are not met there will be no projection created or displayed.

4. Graphical User Interfaces (GUIs)

4.1 Opening Interface (Optional, appearance may vary)



Optional GUI if the lines:

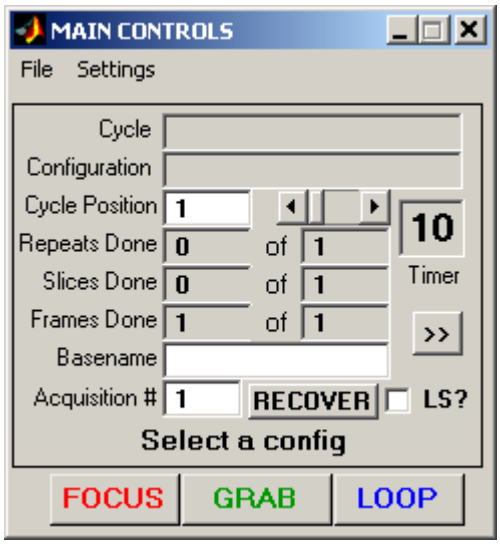
```
global gh
gh.openingGUI = guihandles(openingGUI);
```

are included in startup.m. It enables the user to select *ScanImage*, *ImStack*, or return to the Matlab shell. Clicking the appropriate button does each.

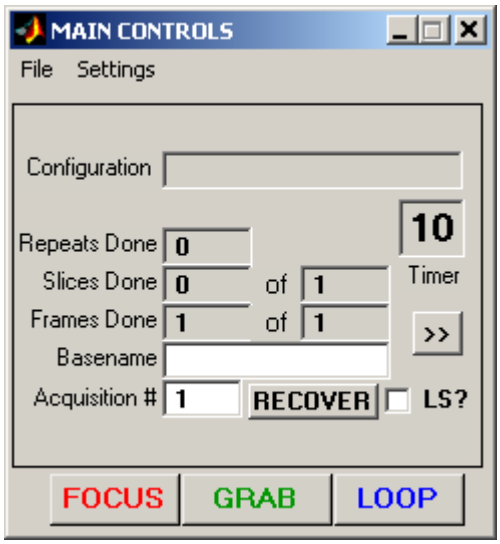
4.2 Main Controls

This is the central user interface and contains a menu bar that opens the other windows. The first view is when the software is in **Cycle** mode, and the second is when it is in **Standard** mode. The differences are described above, but many users will use the **Standard** mode for simple image acquisitions that do not require multiple timed stage positions, as the **Cycle** will allow. The >> button allows the window to be expanded (either mode) to see and edit scan parameters while focusing.

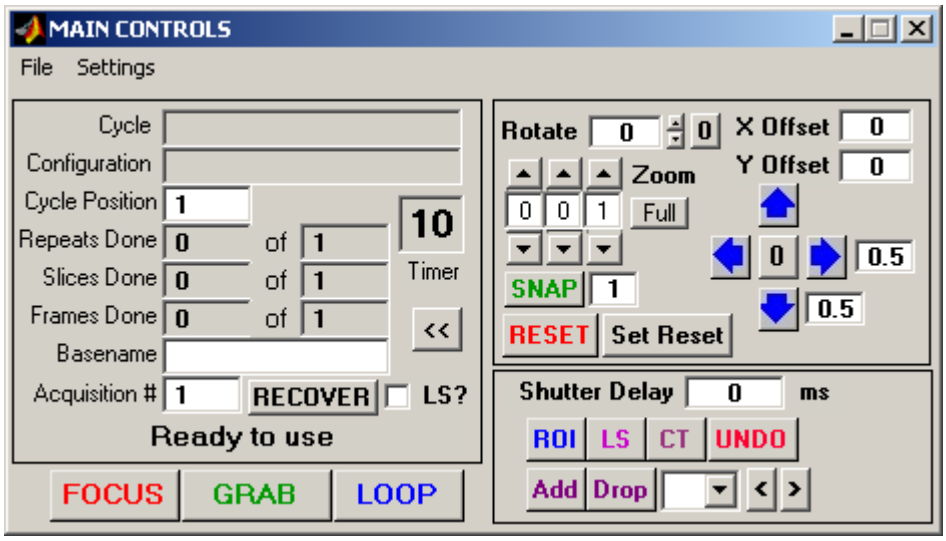
Cycle Mode Display



Standard Mode Display



Scan Parameters



Main Control Window Fields:

Cycle (cycle mode only) – Name of the cycle currently loaded.

Cycle Position (cycle mode only) – The current position of the cycle displayed in the cycle field. This is the cycle position actually dictating the scan parameters when an acquisition begins. The cycle will also start at this position if LOOP is pressed.

.

Configuration – Name of the configuration currently loaded. It changes when the cycle position increments (or decrements) if necessary.

Timer – Keeps a count of the time from the start of acquisition until the next acquisition begins. It is used primarily in cycles where timing is paramount.

Status – String that tells the user what the software is doing and any errors that occurred. In the above diagram, it says "Ready To Use" (**Cycle** mode)

Repeats Done – The number of times the current acquisition has repeated.

Slices Done – The number of different focal planes imaged.

Frames Done – The number of frames acquired in the current acquisition or given focal plane.

Basename – The user defined name that files will be saved as.

Acquisition # – The counter that distinguishes one acquisition from another. Increments automatically for saving purposes, and is appended to the base name and path to complete the save path.

RECOVER – Occasionally the system will error in a way that does return the buttons to their enabled position, or the MP285 motor will error. This button will abort any current acquisition, either Focus, Grab, or Loop and will reconnect to the MP285 motor.

FOCUS

The **FOCUS** button puts the software into a mode suitable for moving the stage and the focus of the scope to locate a structure to image. This mode does not save any data, but rather updates the image in real time in the acquisition windows. If you select image in the **Channel** window, you will automatically be able to focus on that channel as well.

GRAB

The **GRAB** button does a full acquisition determined by the settings the user desires. For instance, if there is only one Z-Slice and 10 frames, the data will be collected for 10 frames in that image plane. If the average button is checked, these 10 images from that plane will be Kalman averaged and only the averaged image will be saved. Similarly, if 10 Z-Slices are selected, 10 frames will be averaged in each plane and the stack will be saved.

The Grab button does one acquisition only.

LOOP

The **LOOP** button controls a series of acquisitions determined either by the current cycle or by the number of repeats selected in the Standard Controls GUI (see below). It essentially loops indefinitely through the cycle acquiring data and saving it to disk. It is aborted by selecting ABORT from the main controls window, which is what the **LOOP** button says when pressed one time.

Scan Parameters Window Fields:

Snap – Acquire a single image, without autosaving or incrementing any of the counters. The textbox next to the button specifies the number of frames to acquire (the frames are automatically averaged).

Rotate – Rotates the current image by the specified angle in degrees.

Zoom – Multiplies the amplitude of the scan by the reciprocal of the zoom factor. The zoom can be changed in unit intervals (left arrow) or tens (right arrow). Zoom of 1 is most zoomed out (largest field of view).

Full – Sets Zoom to 1.

X/Y Offset – Sets the offset voltage applied to the mirrors. Effectively moves the image around in the field of view.

Arrows – Sets the XY offset to move the image in the desired direction. The boxes adjacent to the right and down arrows specify the voltage to jump when the buttons are depressed. The 0 button in the center sets the X/Y offsets to 0.

Reset – Sets all scan parameters to their default values: Zoom =1, Rotation = 0, and X/Y Offset = 0.

LS ? – Toggles between frame scanning and line scanning modes. Sets the Y Amplitude to 0.

Shutter Delay – Sets the time after the beginning of the acquisition to open the shutter. A value of zeros opens the shutter immediately before acquisition begins. It is important to note that the shutter timing is CPU controlled. Because Matlab is single threaded, the shutter may only be opened/closed in between acquistion stripes and when CPU time has become available. This limits the range of useable time settings. For more precise shutter control, custom software/hardware is necessary.

ROI – Selects an ROI (Region Of Interest). It will allow the user to draw a box on the display window, which will then become the designated ROI.

LS – Selects an angle for a linescan. Much like for ROI, a box will be drawn on the display window.

CT – Centers the scan on a selected point. The point is selected by clicking the mouse in the display window.

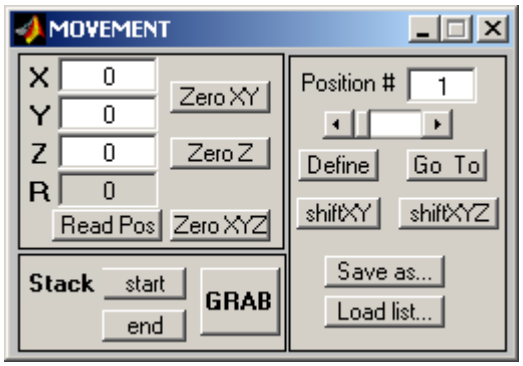
UNDO – Returns the scan to its last position.

Add – Adds an ROI to the list of ROIs.

Drop – Drops an ROI from the list of ROIs.

4.3 Movement

A control the MP-285 motor directly and also defines pre-determined positions for the cycle. The motor is not in constant communication with the software. You may manipulate the MP-285 by the manual controls and it will not update the software until you read the new position into the GUI. However, you may move the motor via the GUI in real time.



Fields:

X,Y, or Z – Current X, Y, or Z relative position. Typing the position here automatically moves the motor to that new position.

R – Current distance from origin (square root of the sum of the squares of X,Y, and Z).

Read Pos – Read the current position of the MP-285 motor into the GUI.

Zero XY – Sets current coordinates of the motor to zero for relative position XY coordinates.

Zero Z - Sets current coordinates of the motor to zero for the Z (focus) position. This is useful when focusing.

Zero XYZ – Sets the current coordinates of all three directions to zero.

Stack Start – Define the current position as the start of a stack acquisition.

Stack End – Define the current position as the end of a stack acquisition.

Position # - The current counter for the positions function. You can define any number (up to 100) of XYZ coordinates as positions, labeling them with the number indicated here. Read below for the details of how to use this feature.

Define – Sets the coordinates of the current point to the position number indicated in the **Position #** field. These can be overwritten at will.

Go To – Will send the coordinates for the new position to the motor and update the GUI.

Shift XY – Will shift all the coordinates stored in the position array by the amount specified in the X and Y fields, leaving the focus (Z) alone).

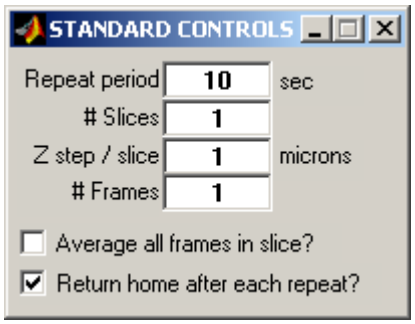
Shift XYZ – Will shift all the coordinates stored in the position array by the amount specified in the X,Y, and Z fields.

Save as… - Save current list of positions as a .mat file.

Load list… - Load a previous positions list into the computer.

4.4 Standard

Contains the most commonly changed acquisition parameters and the repeat length to do the same experiment repeatedly. This window is left open most of the time, and has most of the commonly used parameters easily accessible. It also allows the user to set a **Repeat Period** for doing the given acquisition a number of times.



Fields:

Repeat Period – The time in seconds between acquisitions. The set acquisition will occur at this rate if **LOOP** is pressed by the user in the **Main Controls** window.

Slices– The number of different focal planes to image.

Z Step / Slice – The amount to increment (in microns) between Z-slices. This can be either positive or negative.

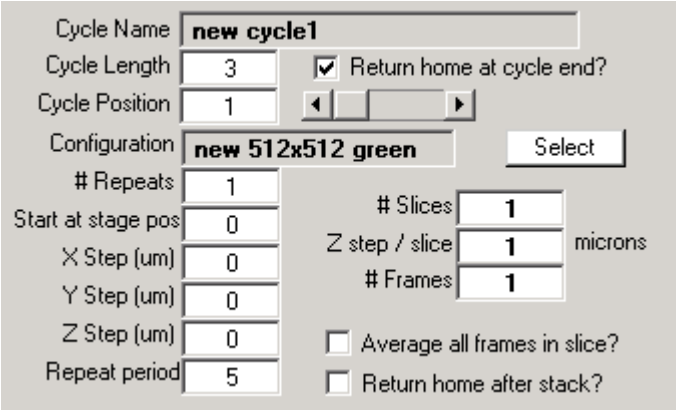
Frames– The number of frames to be acquired in the current acquisition or given focal plane.

Average all frames in Slice? – If checked, the program will Kalman average over the number of frames per slice defined above. This is done for each Z plane.

Return home after each repeat? – Check this if you want to return the position of the stage to the home position in between acquisitions.

4.5 Cycle

Interface that allows the user to predefine a cycle: cycles are timed series of acquisitions where the user specifies the type, length and position of each acquisition. There is no limit to the complexity or length of a cycle.



Fields:

Cycle Name – The name of the currently loaded cycle.

Cycle length – The number of positions defined in the current cycle. This needs to be set to some number greater than 1 in order to define a new cycle if one is not already loaded.

Cycle position – current position in the cycle

Configuration – The configuration the current cycle position will use for an acquisition.

Select – Define a new configuration as the one for the current cycle position.

Repeats – The number of times to loop through the current cycle position before moving to the next position.

Start at Stage pos – Which position from the positions list loaded for the current cycle to start the acquisition.

X, Y, Z step – Move this many microns in the given direction between repeats of the current configuration.

Repeat Period – Time to wait before repeating a given configuration. This is the same as in the **Standard** window.

Slices– The number of different focal planes to image.

Z Step / Slice – The amount to increment (in microns) between Z-slices. This can be either positive or negative.

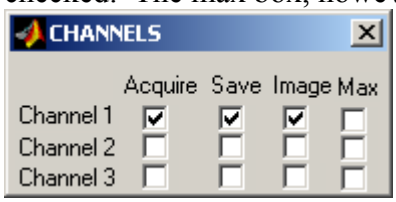
Frames– The number of frames to be acquired in the current acquisition or given focal plane.

Average all frames in Slice? – If checked, the program will Kalman average over the number of frames per slice defined above. This is done for each Z plane.

Return home after each repeat? – Do you want to return the position of the stage to the home position in between acquisitions.

4.6 Channel

This allows the user to select which channels to acquire, save, image, and do running max projections . The check boxes for acquire, image and save automatically become checked when the acquire button is checked. The max box, however, does not update when acquire is checked.



Fields:

All the fields here set parameters for the various hardware channels. If you are doing a normal acquisition, simply check the acquire box next to the appropriate channel.

Acquire – If checked, the system will read in data from this channel.

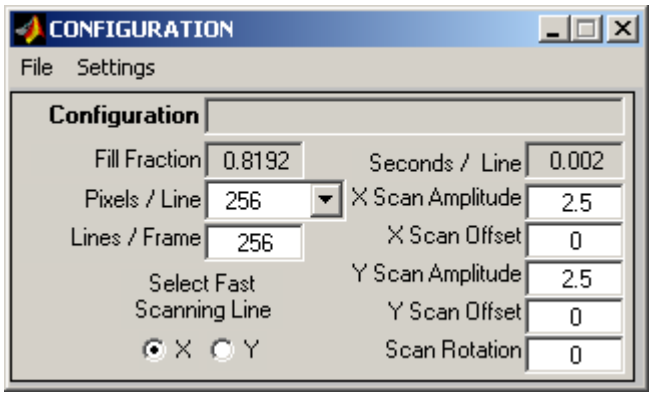
Save – If checked, the acquired data will be logged to disk after each acquisition using the user defined save path and **Base name**.

Image – If checked, the program will display the acquired data. Image will do nothing if there is no acquisition (acquire not selected).

Max – This will compute and display a running max projection of the current channel. This may be selected without Image being selected. However, max is only meaningful if Kalman averaging is on or if the number of frames per slice is one, and it will only be computed if these criteria are met.

4.7 Basic Configuration

Contains basic scan parameters that can be user defined. It also sets which mirror is fast (line) and slow (frame) scanning. The other parameters are accessed through the menu bar by selecting the Advanced pull-down item.



Fields:

Configuration – Name of the current configuration with the displayed acquisition parameters.

Fill Fraction – Defines the fraction of time from the msPerLine that is used for image formation.

Pixels/Line – Select the resolution in the X direction.

Lines / Frame – Sets how many lines to acquire in a given acquisition.

Select Fast Scanning Line – Designates which direction for the fast (high frequency) scanning. The other is automatically set to the slow channel. By selecting one channel and setting the Scan Amplitude (see below) of the other to 0 puts the system in a line scan mode.

Seconds/Line – Displays the actual time that it takes to scan across the sample with the fast channel.

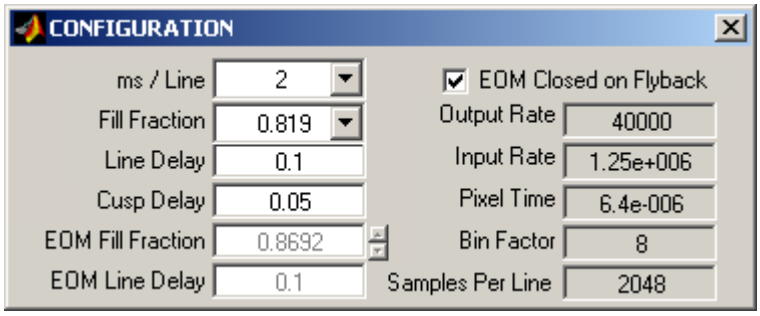
X/Y Scan Amplitude – Amplitude (in Volts) of the mirror oscillations. Larger amplitudes scan more of the sample per pass.

X/Y Scan Offset – Sets the zero of the scan oscillations. Effectively moves the laser around in the field of view.

Scan Rotation – Amount (in degrees) by which you would like to rotate the scan.

4.8 Advanced Configuration

Contains more advanced acquisition parameters, such as **Fill Fraction** and **Line Delay**.



Fields:

ms/Line – Sets the time to scan across the sample in milliseconds. The software will actually set this to a different value depending on other acquisition parameters (like the **Fill Fraction**). The true value is displayed in the **Basic Configuration** field **Seconds/Line**.

Line Delay – The time in seconds to delay using data after the start of a line scan.

Cusp Delay – The time in seconds to extend data acquisition to account for the phase shift of the mirrors when compared to the driving voltage. See **Section 7** for details.

Line Delay – The time in seconds to delay using data after the start of a line scan.

Output Rate – The current DA rate for driving the scan mirrors.

Input Rate – Current rate at which we sample the pixel intensities.

Pixel Time – The calculated time elapsed per pixel in the final image.

Bin Factor – Online processing of data requires adding pixel intensities over this many values to accumulate one pixel.

Samples Per Line – The actual number of samples acquired per line. This is binned according to the Bin Factor to yield the desired Pixels / Line.

EOM Fill Fraction – This works similar to Fill Fraction, but is defined for the Pockels cell signal.

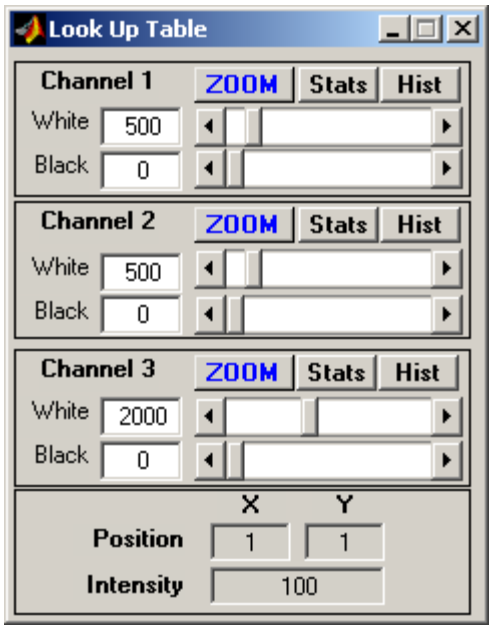
EOM Line Delay – This works similar to Line Delay, but is defined for the Pockels cell signal.

EOM Closed On Flyback – When checked, this automatically sets the EOM Fill Fraction and EOM Line Delay to match the Fill Fraction and Line Delay (respectively), and disables the EOM variables from being edited in the GUI.

4.9 Look Up Table

Allows control of the look up table for the imaging windows. Each channel has its own look up table controlled by this GUI. All the images on a given channel (acquisition, max, or in the current viewer) are

controlled here. This GUI also has a display for the current position and the intensity of a pixel, which is updated simply by clicking on an image at the location you wish to get the intensity and coordinates.



Fields:

White – Any intensity equal to or above this value is displayed as white on the image.

Black – Any intensity equal to or lower than this value is displayed as black in the image window.

Intensity – The display for the intensity of the currently selected point. This is used when you click on a spot on an image.

Current Position X, Y – The current X, Y position of the selected pixel. Used the same as Intensity.

4.10 User Functions

After an acquisition has ended, analysis may be done on the data in a systematic way using User Functions. The user simply selects the directory containing the user functions to use using the Change Path button, and double clicks on the functions to use. This will transfer them from the left list box to the right one.

After execution, *ScanImage* prints out a line stating that the function was executed to the Command Line.

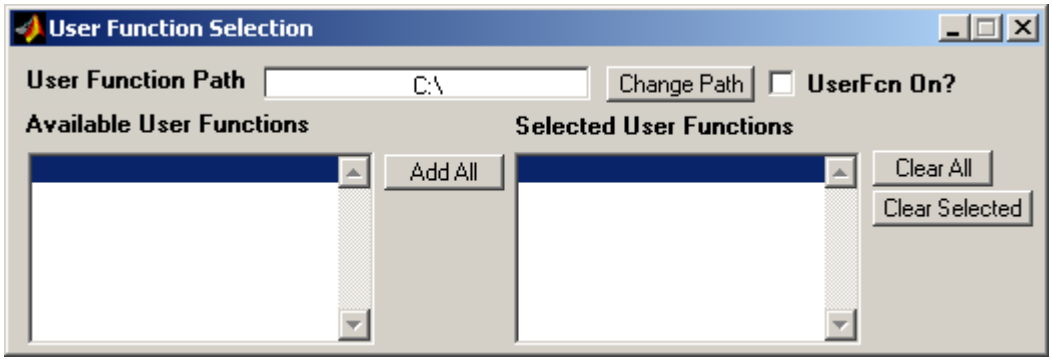
The following are useful variables and functions to program your user functions:

1) Declare `global state` at the top of your function to gain access to the current data in Matlab, including image data, lines per frame, etc…
    a. Please note that in order to do some image processing from data in memory, it may be necessary to check "Keep all slices in memory" from the **Main Controls Settings** menu.

2) Data is stored in the variable `state.acq.acquiredData` as a cell array of numeric arrays, where each channel is indexed by the cell array index.
    a. For example, the data for the second frame of the third channel would be kept in the location `state.acq.acquiredData{3}(:,:,2)`.
    b. In general, the indexing is: `state.acq.acquiredData{channel}(lines, columns, frames)` to get out the appropriate data.

3) The size of the image can be is derived using the `size` command.
    a. Lines per frame = `size(state.acq.acquiredData{channel},1)`.
    b. Pixels Per Line = `size(state.acq.acquiredData{channel},2)`.
    c. Frames = `size(state.acq.acquiredData{channel},3)`.

4) Please see the Matlab manual for details on structure and cell array operations.



Fields:

User Function Path – Path to User functions. The m files in this directory are placed in the Available User Functions list box.

Change Path – Change Path to User Function. Updates User Function Path.

User Function On ? – Executes the User function(s) listed in the Selected User Function list box. After acquisition if checked.

Available User Functions – List of all m files in the directory specified by User Function Path. You can transfer functions from this list to the Selected User Functions list box by double clicking on them.

Selected User Functions – These are the functions that will be executed when the UserFcn On is checked.

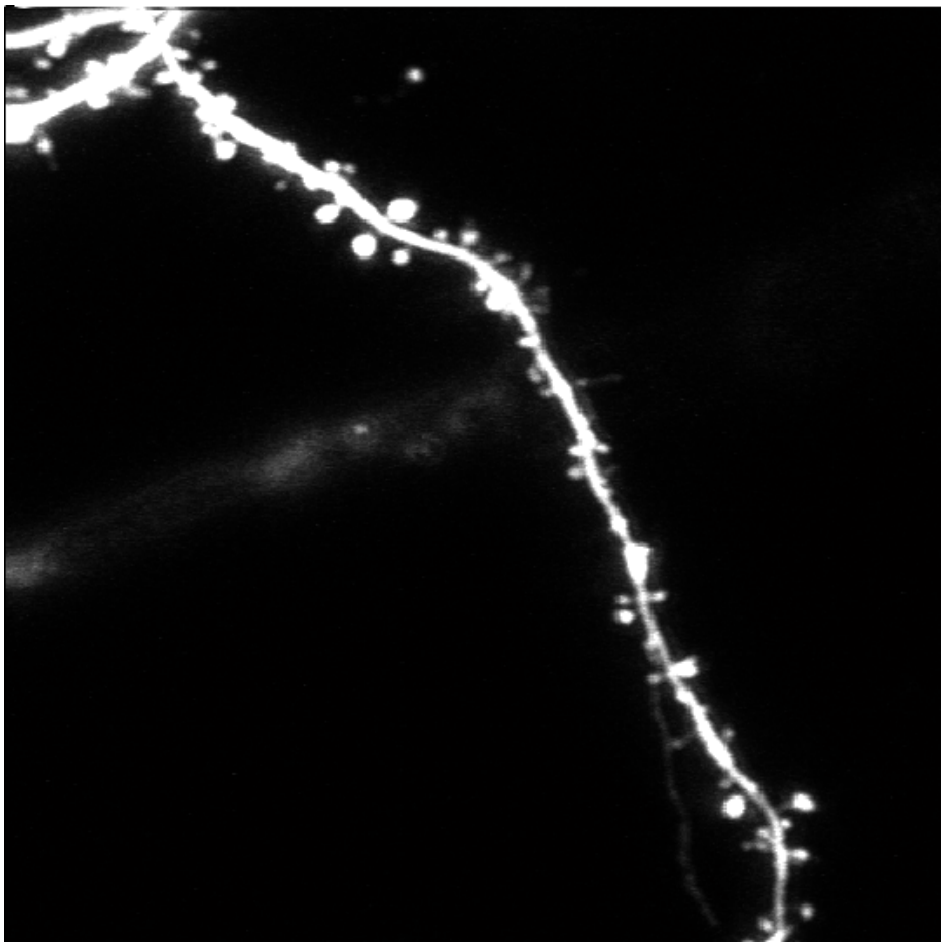
Add All – Adds all the Available User Functions to the Selected User Functions list box.

Clear All – Removes all the Selected User Functions list box.

Clear Selected – Removes selected (highlighted) function from the Selected User Functions list box.

4.11 Figure and Max Windows

These are where the data is displayed. They are active in the sense that clicking on a position in a figure window updates the current position and intensity on the **Look Up Table** window (see above for details).

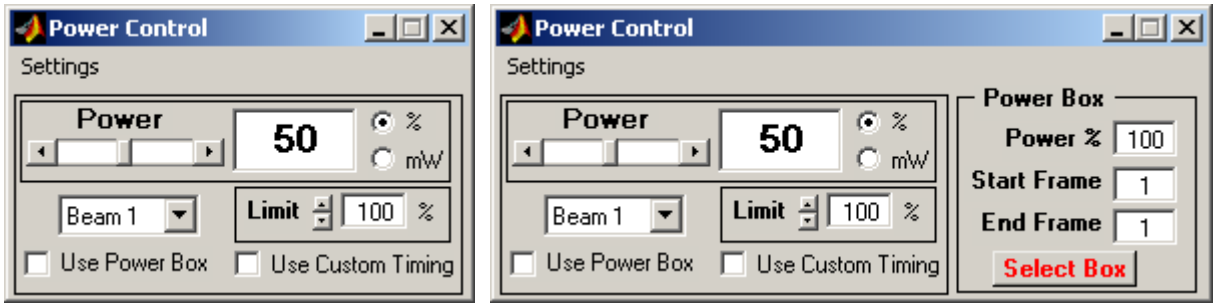


4.12 Power Control

The laser power may be controlled via this software. The Power Control window may be accessed from the Settings menu on the MainControls window. This window, primarily, allows the user to set the power for the scan. The power may be displayed/set in mW (assuming the proper calibration has been established) or as a percentage of available power. The calibration, to allow working in milliwatts,

requires setting the appropriate powerConversionN (where N is a cardinal integer, designating a single beam line) in the standard.ini file. The Power Control window is only enabled when the Pockels cell is activated (pockelsOn is set to 1 in standard.ini).

The Power Control window has a compact and an expanded display. The expanded display, which may be accessed from the Settings menu, includes controls for the Power Box feature.



In addition to expanding the display, the Settings menu allows access to the Power Transitions editor, calibration of the Pockels cell, and viewing of the results of the previous calibration.

Fields:

Power – This field allows the setting of the laser power. The power may be adjusted using either the slider or the edit box.

Units – The units for the power may be changed between percent of maximum and milliwatts, using the provided radio buttons.

Limit – This sets a "soft limit" on the maximum power. It prevents the power field (slider and edit box) from going over a specified value. It is always in the units of percent of available power.

Beam Selector – This popupmenu allows the user to control any of the available beamlines. It also serves as a display, letting the user know which beamline is being worked with at the moment.

Use Power Box – This activates modulation for the selected power box (discussed below).

Use Custom Timing – This activates the use of any defined power transitions (discussed below).

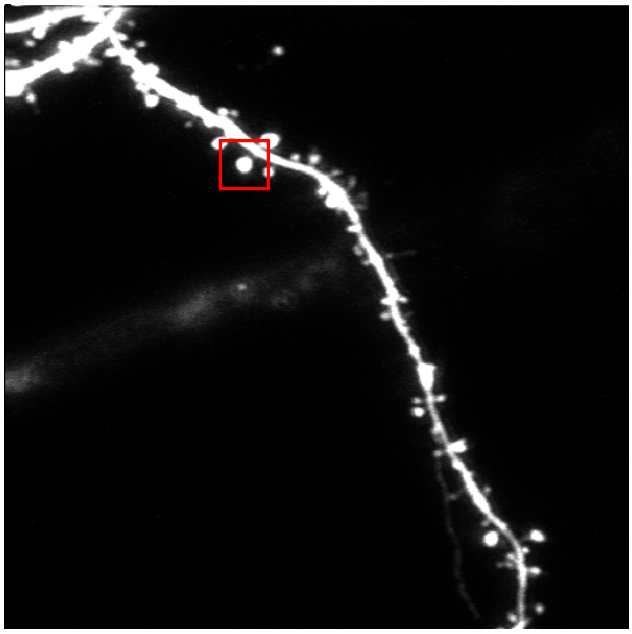
The following fields are within the Power Box portion of the GUI. The power box allows a (necessarily rectilinear) portion of the total scan to have a different power setting than the rest of the scan. This power setting may apply to some or all frames, within a multiframe acquisition.

Power – This defines the power within the selected box, in units of percent of available power. The power outside the box is defined by the normal power field (as discussed above).

Start Frame – This determines at which frame, in a multiframe acquisition, to apply the special modulation to the defined power box. For times not after the start frame, the power will be the same as for the region outside the power box.

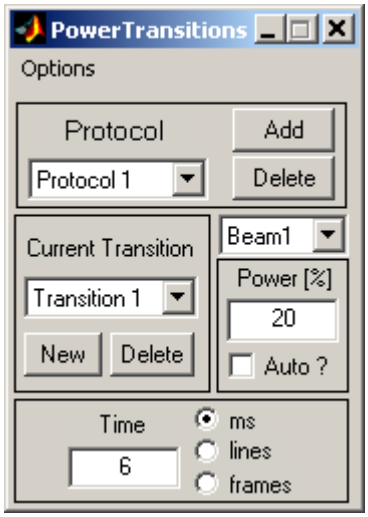
End Frame – This determines at which frame, in a multiframe acquisition, to stop applying the special modulation to the defined power box. For times not before the end frame, the power will be the same as for the region outside the power box.

The power box is only displayed on the acquisition windows when the User Power Box checkbox is checked. A different color is chosen for each beam. The box may be moved or stretched. To move the box, left click on the upper-left corner of it. To stretch it, right click on the upper-left corner of it. An enabled power box would appear as shown below.



4.13 Power Transitions

The Power Transitions editor may be accessed from the Settings menu on the Power Control window or the Settings menu on the Main Controls window. This editor creates an array of transitions in laser power. It allows the user to create transition events (each consisting of a time and power), linked to each beamline. The set of transitions will get stored along with all the other user preferences. This feature is useful for rapid modulation of light intensity (e.g. FRAP and photoactivation).



Fields:

Protocol – Multiple sets of transitions (protocols) may be defined, for different aspects of experiments.

Add – Create a new protocol.

Delete – Delete the current protocol.

Beam – This selects for which beam to view/edit power transitions. Each beam may have its own unique set of transitions.

Current Transition – This popupmenu chooses which transition, in the array of transitions, to view/edit.

New – Creates a new transition, populated with default values, and adds it to the end of the array of transitions.

Delete – Deletes the currently selected transition.

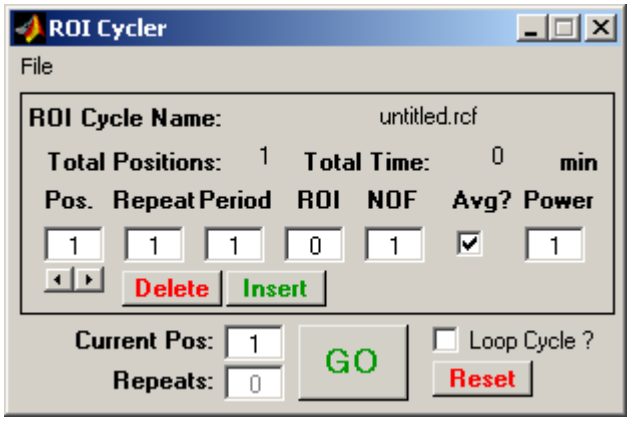
Power – Sets the power in percentage of available power. This power will get applied at the associated time, and will last until the next transition (temporally) or the end of the acquisition.

Auto – By checking this box, the power will automatically go to the power that is currently specified in the Power Control window, at the time of this transition.

Time – This is the time, in the specified units, at which to execute this power transition. The units of time may be selected using the three radio buttons. The time may be specified/displayed in milliseconds, lines, or frames.

4.14 ROI Cycler

The ROI cycler allows the user to define the progression through a series of predefined ROIs. The ROIs may be defined using the MainControls window. The cycler can associate power settings to each of the ROIs and control the timing in between them. Cycles may be saved to and loaded from files, using the File menu. These cycles do not involve any stage motion, they are all optical.



Fields:

Pos – The position in the list of ROIs to execute. Each ROI to be acquired will be linked to one or more positions (i.e. it may be the first and last in the series).

Repeat – The number of times to repeat this position in the list.

Period – The time, in seconds, in between each repeat of this position.

ROI – Which ROI, as defined using MainControls, to use.

NOF – The number of frames to acquire.

Avg – Enable averaging of frames in this position.

Power – The power associated with this position, specified in percent of available power.

Delete – Delete this position from the list.

Insert – Insert a new position into the list.

Current Pos – The current position in the cycle.

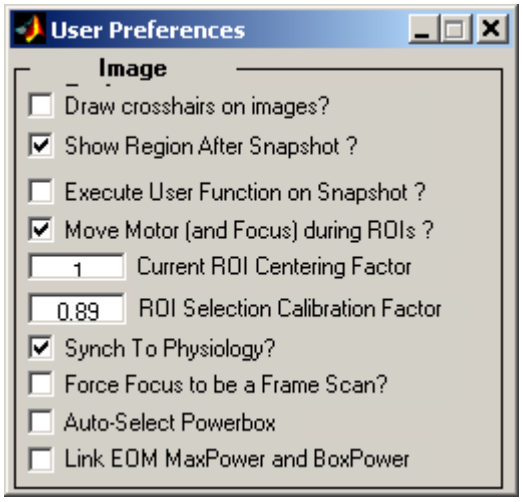
Repeats – The current repeat, within the cycle.

Loop Cycle – Repeat the whole cycle when finished.

The Go button starts acquiring, using the cycle that is currently loaded. The Reset button resets the cycle back to position 1.

4.17 User Preferences

This GUI contains some user-specific preferences about the behavior of ScanImage. They may be accessed by the Edit submenu within the Settings menu on the Main Controls window.



Fields:

Draw crosshairs on images – Place a set of crosshairs on the acquisition windows, to help with orientation.

Show region after snapshot – After a new ROI is acquired, should an image be acquired.

Execute User Function on Snapshot – Toggle the use of user functions when taking Snapshots. In most cases, this should be off, as it would defeat some of the purpose of the Snapshot feature.

Move Motor (and Focus) during ROIs – When switching ROIs, move the motor.

Force Focus to be a Frame Scan – Always do frame scans for Focus, even when in LineScan mode.

Auto-Select Powerbox – If the power box is enabled, without a box selected, a box encompassing the entire first line of the scan will be automatically selected.

5. Designing an Experiment with *ScanImage*

5.1 Acquiring an Image I: Before Illumination

Before an image is acquired, a few things need to be done (especially if it the first time you are using the program).

First, open the various GUIs (from the Main Controls Window) and place them where you like. In particular, select if you are going to be using the more powerful **Cycle Controls** or the **Standard** controls (see the **Main Controls** menu **Settings**) more often. Also, change the number of channels and whether or not to do a running max projection, and see where you would like to place those windows. When every thing is setup the way you like, save the setup by selecting **Save User Settings As…**from the **File** menu in **Main Controls**. This will save the layout of the desktop for future use.

Now open the configuration settings by selecting them from the **Settings** menu in **Main Controls** and make sure they are correct. If not, make any changes that are necessary by updating the GUI fields. If correct, you are ready to save the configuration. You do this by selecting **Save Current Configuration As**…from the **File** menu in **Main Controls**.

Once this is complete, you must type in a **Basename** in **Main Controls** and select a save path for the acquired data. Saving of data in this software is done automatically by selection of the base name, path, and then appending to it the acquisition number (displayed in the **Main Controls**). To choose the save path, select **Set Save Path** from the **File** menu in **Main Controls**.

If you are using a cycle, open **Cycle Controls** by going to the **Settings** menu in **Main Controls** and selecting **Cycle Controls**. This will close the **Standard** window and open the **Cycle Controls** window. Define a cycle of acquisitions and positions (directions for creating a cycle are given below), when complete, you may save the cycle by selecting **Save Current Cycle As**…from the **File** menu in **Main Controls**.

If you are not using a cycle, define the number of slices, the number of frames, whether or not to average, the repeat period, and the Z-step per slice in the **Standard** window.

You can acquire up to three channels simultaneously. You set the channels to acquire by selecting **Channels** from the **Settings** menu in **Main Controls**. By default, when the channels are changed, *ScanImage* computes the dark channel noise for the PMTs (photomultiplier tubes). This PMT offset (essentially the reading of any ambient light, beyond normal noise) is then subtracted from the acquired data automatically. The exception is for channel 3, which always has a PMT offset of 0, since this is the channel used for DIC imaging, where the offset is integral to the image acquisition.

If these steps are done correctly, you are ready to begin acquisition.

5.2 Acquiring an Image II: Data Collection

Once the configuration defaults and cycle are saved, you are ready to begin data acquisition. There are three buttons located on the Main Controls GUI labeled FOCUS, GRAB, and LOOP. These are detailed below.

FOCUS

The **FOCUS** button puts the software into a mode suitable for moving the stage and the focus of the scope to locate a structure to image. This mode does not save any data, but rather updates the image in real time in the acquisition windows. If you select image in the **Channel** window, you will automatically be able to focus on that channel as well.

GRAB

The **GRAB** button does a full acquisition determined by the settings the user desires. For instance, if there is only one Z-Slice and 10 frames, the data will be collected for 10 frames in that image plane. If the average button is checked, these 10 images from that plane will be Kalman averaged and only the averaged image will be saved. Similarly, if 10 Z-Slices are selected, 10 frames will be averaged in each plane and the stack will be saved.

The Grab button does one acquisition only.

LOOP

The **LOOP** button controls a series of acquisitions determined either by the current cycle or by the number of repeats selected in the **Standard** window. It essentially loops indefinitely through the cycle acquiring data and saving it to disk. It is aborted by selecting ABORT from the main controls window, which is what the **LOOP** button says after it is pressed once.

Depending on the acquisition mode you selected, GRAB or LOOP will start an acquisition.

5.3 Acquisition Modes

Standard Acquisition Mode

In standard mode, the program runs much like the confoc or flouview interface. The stage position, basic scanning parameters, averaging, and maximal projections can be selected for each channel and changed rapidly. Many experienced users will find this mode extremely familiar and straightforward.

This mode allows the timed repeat of experiments and also allows most of the motor functions to be accessed directly. If you wish to do a single acquisition, simply press GRAB. If you want to repeat the acquisition a number of times, update the **Repeat Period** in the **Standard** window and press **LOOP**.

Cycle Mode

This is the more complex mode where the user may predefine positions and configurations to run in series. The user focuses and finds a list of positions on the sample they would like to image, and then they define a cycle through those positions matching up with scan configurations defined for those positions.

Cycle controls can be changed and displayed without the actual acquisition parameters being changed. The acquisition parameters only get changed if the **Cycle Position** in the **Main Controls** window is changed, not when it is updated in the **Cycle Controls** window. This allows the user flexibility in defining cycles independently of changing acquisitions parameters on the current configuration. To display a cycle, select **Display Cycle** from the **File** menu in **Main Controls**.

Creating a Cycle

1) Be sure that **Cycle Controls** is open by selecting it from the **Settings** menu in the **Main Controls** window.
2) Change the field for **Cycle Length** to match the number of different positions your cycle will have.
    a. You will notice that the **Cycle Name** is not able to be changed. It is updated when you choose **Save Cycle As…** from the **File** menu in **Main Controls**.
3) Set the **Cycle Position** to one and input into **Cycle Controls** the parameters for position one of your cycle. The definitions of each field are given **Section 4.**
4) When done with the first position in the cycle, set the **Cycle Position** to two and repeat Step 3.
5) When you are done, choose **Save Cycle As…** from the **File** menu in **Main Controls**.
    a. This will update the **Configuration** and **Cycle Name** in the **Cycle Controls** window.

6. Signal Conditioning

One of the key advantages of *ScanImage* over other laser scanning systems is that it does not require elaborate signal conditioning electronics. In particular, other laser scanning systems require that photocurrents are integrated over the duration of a pixel time electronically. Instead *ScanImage* acquires fluorescence time data at a very rapid rate (1.25 MHz) and then passes the task of integration to the CPU of the computer. Using a modern PC real-time digital integration can easily be achieved even using a high-level programming environment such as Matlab.

The only signal conditioning requirement is that the PMT currents are converted to voltages and low pass filtered around 300-500 kHz. This can be achieved using custom-made amplifiers assembled for cents (for example see Horowitz, P., and Hill, W. (1989). The Art of Electronics, 2nd edition (Cambridge, Cambridge University Press)) or using a commercial product (SR570, Stanford Research Systems). For best results set the gain of the amplifiers to get several pixel values (5-10) per photon; for example, for Hamamatsu R3896 PMTs running at 800V, a gain of 10 μA/V would be appropriate.

The analog output sampling rate (for determining the mirror driving functions) is set to 40 kHz and can be fed directly to mirror drivers. The output and input rates are set in the .ini file and should not be changed from their default values (40kHz, 1.25MHz) for optimal performance.

The input sampling is over-sampled by the amount set in the **Bin Factor** field in the **Advanced Configuration** window. For example, if you want 512 **Pixels Per Line**, *ScanImage* acquires that number of pixels times the **Bin Factor**. This is called the **Samples Acquired Per Line** and can be seen in the **Advanced Configuration** window. The correct **Pixels Per Line** is achieved by adding the acquired intensities over the number of samples equal to the **Bin Factor**, and setting this sum equal to the intensity of the pixel in the image.

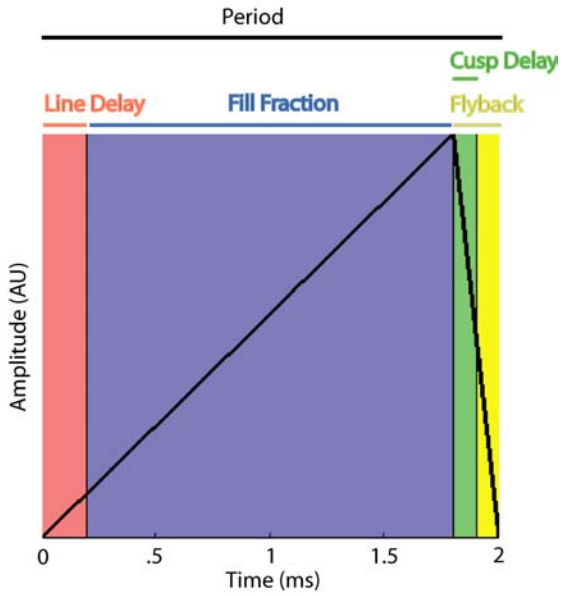
Even more samples than this are acquired, since we do not use some portion of the acquired data because the mirrors are not yet positioned correctly for image acquisition. In each output cycle, the mirror scans across the sample and then has to "flyback" across the sample to prepare for the next line of acquisition.

The amount of time it takes to flyback is set in *ScanImage* by 2 other acquisition parameters: the **Fill Fraction** and the **Line Delay. Fill Fraction** (φ) is a parameter you can set from the **Advanced Configuration** window, and it in turn sets the flyback time. The **Fill Fraction** (f) is the fraction of time the mirrors are actually acquiring data from a given line scan, at all other times during the scan, no data is collected. If the line scan takes 2 ms with a **Fill Fraction** of .8, 1.6 ms of data is used to create an image every line. The **Line Delay** (δ) is the time in seconds to wait before collecting data for the next line in order that the mirrors may adjust from their own inertia. The default value is .2 seconds. All of these quantities are related by the equation:

$$FillFraction * msPerLine = msPerLine - LineDelay - FlybackTime$$
$$\phi\tau = \tau - \delta - f$$

where **msPerLine** (τ) is the amount of time it takes to acquire a single line of data. This can be updated in the **Advanced Configuration** window. A diagram of the definitions is shown below.

The acquisition is timed to be simultaneous with the output to the scanning mirrors via a digital trigger. This TTL pulse is delivered from the NI 6110E board itself and the wiring schematic is given in **Section 2**.



The data acquired from the NI board is in 'Native' format and is saved into uint16 arrays after processing. The timing of the input is achieved by ensuring that the **Input Rate**, **Fill Fraction**, **Samples Acquired Per Line**, and the **ms Per Line** variable are related as follows:

$$msPerLine * InputRate = SamplesAcquiredPerLine / FillFraction$$

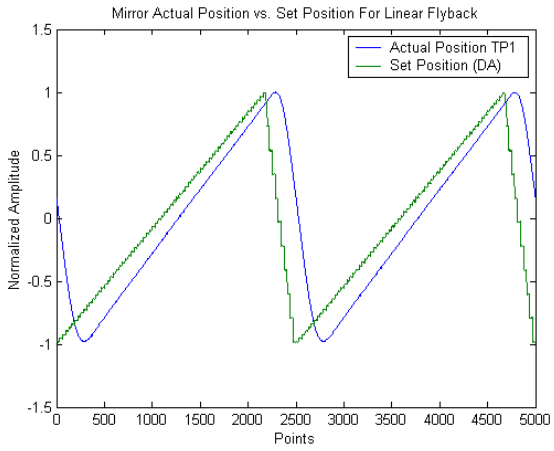
where the left hand side is the number of samples that need to be acquired in order to have **Samples Acquired Per Line** available for binning into the final pixel intensities (as described above). This equation is fundamental to bring accurate acquisitions to fruition. For a fixed **Input Rate** and **Samples Acquired Per Line** (these are fixed in *ScanImage*), only certain pairs of values for the **ms Per Line** and **Fill Fraction** will satisfy the above relation. These have been hard coded into the program, and are listed below for reference:

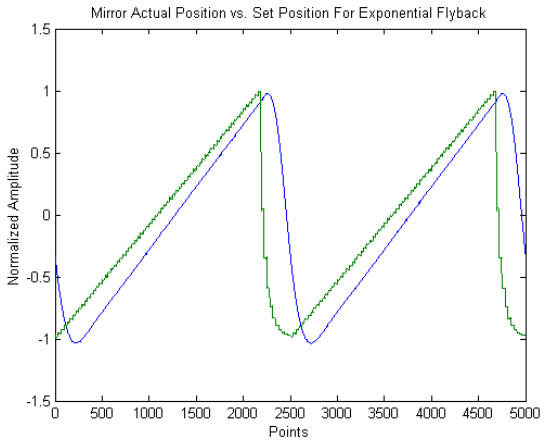
Set ms Per Line	Actual ms Per Line	Fill Fraction	Samples Acquired Per Line
1	1.15	0.71234782608696	1024
	1.1	0.74472727272727	1024
	1.05	0.78019047619048	1024
	1	0.81920000000000	1024
2	2.3	0.71234782608696	2048
	2.2	0.74472727272727	2048
	2.1	0.78019047619048	2048
	2	0.81920000000000	2048
	1.9	0.86231578947368	2048
	1.8	0.91022222222222	2048

	4.6	0.71234782608696	4096
	4.4	0.74472727272727	4096
	4.2	0.78019047619048	4096
4	4	0.81920000000000	4096
	3.8	0.86231578947368	4096
	3.6	0.91022222222222	4096
	3.4	0.96376470588235	4096
	9.2	0.71234782608696	8192
	8.8	0.74472727272727	8192
	8.4	0.78019047619048	8192
8	8	0.81920000000000	8192
	7.6	0.86231578947368	8192
	7.2	0.91022222222222	8192
	6.8	0.96376470588235	8192

The **Set ms Per Line** is the value the user sets in the **Advanced Acquisition** window, and *ScanImage* automatically chooses the **Actual ms Per Line** based on the user selected **Fill Fraction**. Note that for a **Fill Fraction** of .8192, the **Set** and **Actual ms Per Line** are identical.

As can be seen below, there is a phase difference of about 100 μsec between the voltage driving the mirrors and the actual mirror position. To compensate for that, a **cusp delay** (κ) has been included. This will allow the data acquisition to continue beyond the command voltage cusp, allowing more useful data to be collected. This parameter is found in the **Advanced Configuration Window**. Changing the shape of the flyback does not alter the time course of the mirror position significantly.





7. General Logic of *ScanImage* from a programming perspective

The program uses a number of text files described above to set various parameters for the system simultaneously. These files are also parsed and create structured arrays called state and gh. Structured arrays are like the normal tree structures you use in windows to store data. For instance the path C:\matlabR12\work\file specifies the location of file as being in the folder work in the folder matlabR12 that is on the C drive. In structure terminology, this would be written as c.matlabR12.work, since structures use periods and not back slashes as delimiters. If you type state at the command line, you bring up the first level of state, which you can then move down the hierarchy in by selecting state.acq to bring up the acquisition parameters. Finally, if you enter **state.acq.numberOfFrames**, Matlab will return the number of frames in the current acquisition. This is where the data is stored in the state structure.

The GUIs interact with the state structure in the following way: each field in a GUI that you change correspondingly changes the value of a state variable (the converse is not true, however), and the program looks to the state structure to determine what it should do. For example, the checkbox in the **Cycle** or **Standard** window that says **Average?** sets a flag telling the software whether or not to Kalman average all the slices in a given focal plane. It actually sets the value of the state.acq.averaging variable to 1 (checked) or 0 (unchecked). The software then uses the information stored in state to accomplish the given command.

This previously described interaction between a global structure array and the fields in Matlab GUIs is generic. It provides an amazingly simple way to create and utilize the awesome graphics and numerical tools Matlab has to offer. The general methodology for creating and maintaining a GUI in Matlab is sketched below.

7.1 Creating a Matlab Graphical User Interface

1) Type `guide` from the command line to open the Matlab graphical user interface builder. A nice introduction to using GUIDE for designing GUIs can be found in the Matlab help directory by selecting **Full Product Family Help** from the **Help** menu. Type `Creating Graphical User Interfaces` into the **Search** field and select it from the list.

    a. Build your GUI by adding buttons, text fields, menu options, etc….

    b. Be sure to give all editable or updated fields a **tag**. This should be a descriptive name that will also be the name of a structure field.

        i. You can assign **tag** or declare any other property of a GUI filed by using the **Property Inspector** button located on the GUIDE toolbar.

    c. When you are done, save the GUI by selecting Save format The **File** menu**.**

        i. Saving automatically saves the layout as a .FIG file and creates a .M file with the same name. The .M file is where the *Callback* functions for all the buttons you created are located.

1. Menu item callbacks are actually coded directly into GUIDE via the **Menu** Editor**.**

2) Once the .M file is created, a text file listing the name of your structure should be created. This file will exactly resemble a .INI file. The names of the fields and some appropriate initial value should be selected. A structure begins and ends with the declarations `structure` and `endstructure`, respectively. For example:

```
structure example
   structure secondfield
            variableWithSameNameAsGUITag=1 Gui guih.GuiName.GUIField
            variableWithSameNameAsGUITag2=2 Gui guih.GuiName.GUIField2
   endstructure
endstructure
```

will create a structure called `example` with a subfield called `secondfield` and with 2 end fields called `variableWithSameNameAsGUITag` & `variableWithSameNameAsGUITag2`.

These are written as `example.secondfield.variableWithSameNameAsGUITag` and `example.secondfield.variableWithSameNameAsGUITag2`. The values initially set for these variables are 1 and 2, respectively, as indicated in the text file.

   a. Although not necessary, it is vastly easier if the fields of your GUI and the names of your variables in the structure are identical.

      i. This implies that `variableWithSameNameAsGUITag` would have the same name as `GUIField`.

3) The text next to the variable definition (e.g. `Gui guih.GuiName.GUIField`) describes what GUI and what field of that GUI are connected to that structure field.

   a. Many GUI fields can be connected to one structure field, but the converse is not true.
      i. To add many Gui fields to one structure variable, just repeat the text (including `Gui`) with the new name (e.g. `Gui newName`).

   b. The actual GUI fields can be extracted from the GUI .M filename using the Matlab function `guihandles`. This function takes the filename of the GUI as an input, and it returns a structure containing the **tags** for all the fields in the GUI. This is why it is *very important* to fill in the GUI **tag** property when making the GUI.

      i. `Guihandles` also opens the GUI figure.

   c. Thus, you create *another* structure array containing the GUI tags, and these are what are input in the text file (see **a** above).

   d. The start up script for your GUI will contain the following line which creates the GUI structure:

```
guih.GuiName = guihandles(GuiName)
```

4) Once the text file is created, it should be saved with the extension .INI. The next line of the start-up for your GUI will be to initialize the GUI you created using the `initGUIs` function. This function will actually create the variable structure and tie the designated GUI fields to that structure.

```
initGUIs(name.ini);
```

5) If successful, all of the GUI fields will be updated with the initial values and tied to the correct GUIs.
    a. Other text fields can also be set in the .INI file and follow the `Gui guih.GuiName.GUIField` text. See list below:

        i. `Numeric 1`
            1. Ensures field is always a numeric vale.

        ii. `Min/Max Value`
            1. Sets the **min** and **max** values for the field. Useful for sliders.

        iii. `Callback FunctionName`
            1. Sets a function that is executed whenever the GUI is updated.
            2. This is a very nice way to interact with *ScanImage* without changing the code.

6) The .M file for the GUI can now be updated.

    a. In order for the automatic updating described above to work, the `genericCallback(h)` function must be included in the callback.

    b. Now, when you update a GUI field the structure value is also updated.

    c. Add other functions to the callback as necessary.

    d. You are now ready to use the GUI.

7) Other Useful Hints:

    a. When a GUI is updated so is its structure variable, but the converse is not true. Updating a structure variable will not update the GUI automatically.

        i. This is accomplished by the function `updateGUIByGlobal ('StructureVariable.NameAs.String');`.

    b. If you are using editable text boxes, set the **min** to 0 and the **max** to 1. Then use sliders to change the values with min and max set normally. If you do not do this,

when the user enters a value into the text box and hits enter, the cursor will move to the next row of text, not enter the value and execute the callback.

c. If you receive an error that the sliders are out of range, go into **GUIDE** and set their **Value** to one that is in the **min**/**max** range.

7.2 Example: Creating a Matlab Serial Port GUI

Addition of a serial device is easily integrated into the existing framework using the code developed for **ScanImage**. A GUI can be created that can interact with all accessible parameters in **ScanImage** so that custom data exchanging software is not necessary (as is the case for communication between multiple windows programs).

A serial device object can be created and can control devices timed with **ScanImage** or independent of **ScanImage** depending on the device and the user's needs. Control of shutters, Acoustic Optical Deflectors, and even digital cameras is possible via serial ports, and use of these in the same workspace as the 2PLSM software offers many advantages over manual integration of compiled code.

Matlab allows full control over serial pots via their serial port class. You can:

- Configure serial port communications
- Use serial port control pins
- Write and read data
- Use events and callbacks
- Record information to disk

The serial port object created is concise and can be easily managed via standard handle/object Matlab declarations, like `set` and `get`.

Below we will create a small GUI that opens a serial port connection on a specified COM port. The GUI will have 2 buttons: one to connect/disconnect to the port and another to select the port to be used for communication. We can then check the state variable to see the object status and manipulate this object for more advanced communications.

The files for this example are included under the **ScanImage\Documentation** folder in the **SerialPortGUI** directory.

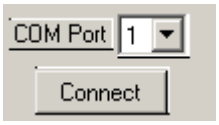
Setting up the serial port INI file

First we need to create an INI file for the GUI. Each Button on the GUI should have a variable in the INI file. The INI file should contain a structure as follows:

```
structure state
    structure serialPort
            objectHandle      % Future handle of the serial Port Object
            comport=1   Gui gh.serialPortGUI.comport  Numeric 1   % COM Port
            connect=1   Gui gh.serialPortGUI.connect  Numeric 1   % Connection Status
    endstructure
endstructure
```

This code will get processed by the `openini` function to create the `state` variable for interaction with the GUI. Save this file as serialport.ini in a folder on the Matlab path.

Once done, type guide to create the GUI. It should be saved in this folder with the name SerialPortGUI. Each of the tag fields in the GUI must correspond to the ones listed in the INI file. For example, the tag field for the COM Port popupmenu should be set to `comport`. When completed, the GUI should appear as follows:



When you save the GUI it will automatically append .FIG to the filename and create a file with the same name but with the .M extension (M file). The M file created is where the callbacks for the GUI buttons are located. They should appear as follows:

```
% --------------------------------------------------------------------
function varargout = connect_Callback(h, eventdata, handles, varargin)
genericCallback(h);
global state gh
if isempty(state.serialPort.objectHandle) % Object Not Initialized
       serialPortGUI('comport');     % Call the comport Callback below
end
if state.serialPort.connect %Connect to Serial Port
       fopen(state.serialPort.objectHandle);
       set(h,'String','Disconnect'); %Change call to Disconnect
else % Disconnect from serial port
       fclose(state.serialPort.objectHandle);
       set(h,'String','Connect'); %Change call to Connect

end


% --------------------------------------------------------------------
function varargout = comport_Callback(h, eventdata, handles, varargin)
genericCallback(h);
global gh state
string=get(h,'String');
val=get(h,'Value');
if isempty(state.serialPort.objectHandle)    % Object Not Initialized
       state.serialPort.objectHandle=serial(['COM' string{val}]);  % Define Object
else
       set(state.serialPort.objectHandle,'Port',['COM' string{val}]); % Update property
end
```

Now when you press connect or change the port it will initialize the serial port object with a call to the constructor `serial`. When the connect button is depressed, the channel is opened for the object on that channel and the button string is changed to disconnect. Depressing this button again will close the connection and set the string back to Connect. If you change the COM port, the object property Port is set to the new channel.

We now will create the startup function for the GUI. We call it startserialPortGUI and it reads simply:

```
function startserialPortGUI
global gh state
% Starts Serial Port GUI
gh.serialPortGUI=guihandles(serialPortGUI);  %Create GUI
openini('serialport.ini');           %Initialize GUIs
```

It only calls 2 functions: `guihandles` to construct the GUI and `openini` to read the INI file and connect the GUI to the `state` variable. Once complete, the GUI is operational.

The call to `openini` in the startup function creates the state variable while the call to guihandles creates the `gh` variable. The `gh` variable contains the handles to the buttons on the GUI, while `state` contains any field you placed in the INI file. When the GUI is changed, the `state` variable is updated also.

We can look at `state` and `gh` by typing them at the command line:

```
>> global state
>> state.serialPort

ans =

      objectHandle: [1x1 serial]
           comport: 2
    globalGUIPairs: [1x1 struct]
           connect: 0


>> global gh
>> gh.serialPortGUI

ans =

    figure1: 99.0015
      text1: 4.0020
    comport: 3.0038
    connect: 100.0017
```

We see that the variable `state.serialPort.objectHandle` contains a handle to a serial object. We can get the properties of this object by typing `get(state.serialPort.objectHandle)`:

```
>> get(state.serialPort.objectHandle)
    ByteOrder = littleEndian
    BytesAvailable = 0
    BytesAvailableFcn =
    BytesAvailableFcnCount = 48
    BytesAvailableFcnMode = terminator
    BytesToOutput = 0
    ErrorFcn =
    InputBufferSize = 512
    Name = Serial-COM2
    OutputBufferSize = 512
    OutputEmptyFcn =
    RecordDetail = compact
    RecordMode = overwrite
    RecordName = record.txt
    RecordStatus = off
    Status = closed
    Tag =
    Timeout = 10
    TimerFcn =
    TimerPeriod = 1
    TransferStatus = idle
    Type = serial
    UserData = []
    ValuesReceived = 0
    ValuesSent = 0

    SERIAL specific properties:
    BaudRate = 9600
    BreakInterruptFcn =
    DataBits = 8
    DataTerminalReady = on
    FlowControl = none
    Parity = none
```

```
PinStatus = [1x1 struct]
PinStatusFcn =
Port = COM2
ReadAsyncMode = continuous
RequestToSend = on
StopBits = 1
Terminator = LF
```

Properties of the serial port object can be changed through the state variable and additional GUI buttons as desired. Sophisticated timing and I/O is easily included in this way.

This example creates a Serial port object and can be expanded to update any of its properties, as well as save and send data using the Matlab serial class. It is one example of the many different features that can be incorporated into **ScanImage** using the modular code developed herein. The power of combining Matlab classes with graphical interfaces in the framework of **ScanImage** is evidenced by the concise way additional components can be created and manipulated with this system.

Appendix A
File Hierarchy and Descriptions

The following is a diagram of the tree structure used for the software:

Level 1	Level 2	Level 3
ScanImage		
	guis	
		flow_control
		gui_building
		gui_callbacks
		gui_handling
	init_files	
	io	
		ad_da
		data_in
		data_out
	file_io	
		configurations
		cycles
		data
		initialization
		positions
		users
	motors	
		MP-285
	documentation	SerialPortGUI
	utilities	
	eom	

ScanImage

Directory that contains the files needed for 2-photon laser scanning microscopy acquisition and output.

guis

Contains both the specific files that create the layouts of the windows and figures and the generic code that simplifies the interaction of the GUIs with Matlab functions. It also contains several "Callback" functions. Callbacks are functions that get executed when a field of a GUI is updated.

flow_control

Contains the files responsible for looping through configurations and the coordination of acquisitions.

gui_building

The actual .fig files that layout the GUIs and their associated .m files are here. The function makeImageFigures is included here, as it is responsible for creating the image windows.

gui_callbacks

Any function that is triggered by the update of a filed in a GUI is here. Also included are secondary callback functions. These are functions that update dependent parameters when one variable is updated and not another. Many of these secondary callbacks execute mathematical operations and determine the value of other variables. For example, when the user updates the msPerLine (milliseconds per line) variable, the pixel time is recalculated and updated also.

gui_handling

This directory has the files that parse the .ini, .usr, .cyc, & .cfg files and update the GUIs from that data.

init_files

This contains the .ini files. These files define the entire state variable with which the program interacts. These files are computer specific and are not meant to be updated regularly. It contains information regarding the tools used on a particular microscope (i.e. motor) .

.ini files are text files and are parsed by functions contained in the file_io\initialization directory to enable communication between Matlab functions and the GUIs. The logic is like other schemes in Matlab: there are parameter-value pairs that are parsed by the software. For instance, if you look at the standard.ini file, you will see a tree like (tabbed-in) structure and after a variable definition (like numberOfFrames) there is an "=" and then a value. This is the initial value of that variable and hence its corresponding filed in a one or more GUIs. Following the variable definition and initial value are a series of other strings (e.g. GUI, Numeric, Callback, …). These are parameters that tell the software to tie the given variable to the value that follows the parameter definition. For example, the parameter GUI is followed by the handle to a particular field in a GUI. When parsed, it will automatically update the state variable whenever the field in the GUI is changed.

The Callback parameter is followed by a function in the Matlab path. Whenever the corresponding variable is updated in a GUI, the function defined here will be executed. This provides a way for users to interface with the software without interfering with its core functions.

io

This directory contains the software that interacts with the NIDAQ board and is responsible for the data acquisition and the driving of the scan mirrors. It also contains the online processing tools, including binning, averaging, and maximal projection functions.

ad_da

Contains the files responsible for setting up the data acquisition objects. This includes making and updating the output to the scanning mirrors and the acquisition of pixel values from the PMTs. Also included are the commands for the shutter switch, the PMT offsets, and the focusing controls.

data_in

Files that manipulate the acquired data and format it into an image. These include the action functions that are paramount for real time data acquisition and display.

data_out

These file create and output the scanning mirror signals.

eom

This directory contains the files related to use of a Pockels cell, for modulating laser power.

file_io

This directory contains the software for the opening, closing, and saving of various parameters and files in the software. The .ini, .cfg, .usr, and .cyc parsing, loading and saving functions are here, as are the functions that write the acquired data to disk. The functions that write the header into the image and parse that header are included, as are the functions controlling the positions the user defines during a cycle.

configurations

The file input and output functions for .cfg files. This includes loading .cfg text files into the workspace and creating .cfg text files based on the current workspace state.

cycles

The file input and output functions for .cyc files. This includes loading .cyc (.mat files) into the workspace and creating .cyc (.mat files) based on the current workspace state. Also included is the code for changing cycle positions in dynamic memory.

data

The input and output functions for the image header construction and parsing. The header is a string that contains all the relevant acquisition parameters in a parameter value format, using carriage returns as delimiters. The functions that read the header (or parts of it) from .tiff files are included here.

initialization

The file input and output functions for .ini files. This includes loading .ini text files into the workspace and creating .ini text files based on the current workspace state.

positions

The file input and output functions for position files. This includes loading position files into the workspace and creating position .mat files based on the current workspace state.

users

The file input and output functions for .usr files. This includes loading .usr text files into the workspace and creating .usr text files based on the current workspace state. The .usr file stores the positions of the GUIs and figures on the screen.

motor

The software makes use of a motorized stage in order to locate object for imaging and, more importantly, for focusing up and down on a specimen. These are the functions that configure and interact with the motor controlling these motions. The Sutter MP-285 is the primary one used, and many functions are written specifically for this device. These include functions that set, query, and zero the motor position.

documentation

This file contains known bugs (Matlab and software specific), directions, and more details on the setting of acquisition parameters.

utilities

General functions that would be useful in many applications in Matlab. This is not an exclusive set, as many of the functions used in the software are also general in their implementation.

Appendix B
Useful Matlab Help and Editing Tools

`help filename`

Typing this at the command line brings up the help file associated with a given Matlab script. The help file is the commented section of a function that appears under the function declaration.

`edit filename`

Typing this at the command line opens the function source code up for viewing and editing.

`which filename`

Typing this at the command line returns the path in which the file is stored.

`daqfind`

Displays all of the currently loaded data acquisition devices.

`get(objecthandle)`

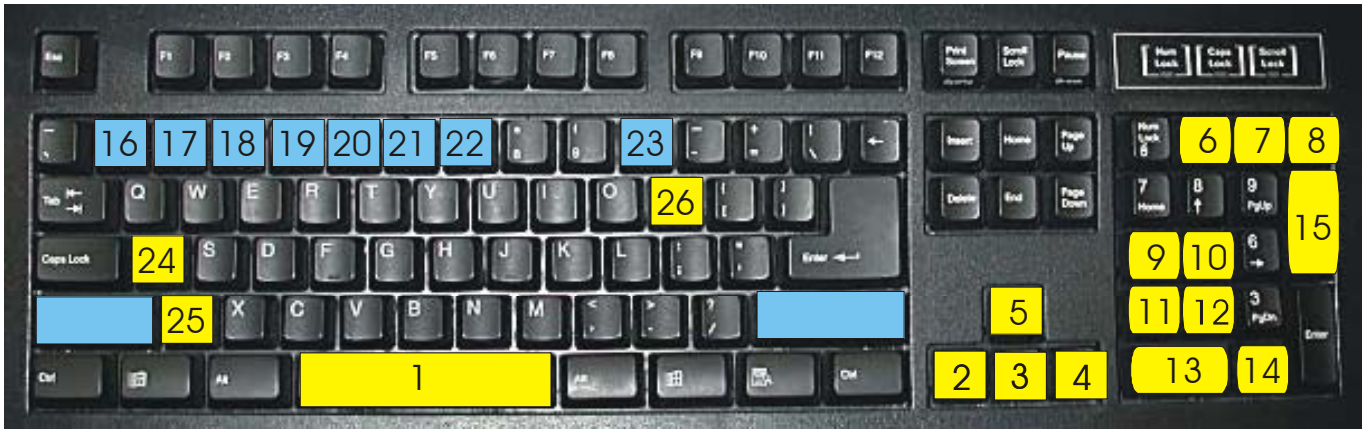
Displays the properties of the object with handle objecthandle.

`clc`

Clears the command window without erasing any variables.

Appendix C
Character Mapping and Shortcuts

ScanImage uses the keyboard for quickly changing scan parameters and configurations. Listed below are the preset buttons mapped in *ScanImage*. Users may assign other buttons as they wish by editing the genericKeyPressFcn in the *ScanImage* directory.



1) Toggle Line Scan/Frame Scan
2) Move Image Left
3) Move Image Down
4) Move Image Right
5) Move Image Up
6) Quick Configuration 1
7) Quick Configuration 2
8) Quick Configuration 3
9) Rotate Image Counterclockwise
10) Rotate Image Clockwise
11) Zoom Out
12) Zoom In
13) Start/Stop Focus
14) Start/Stop Grab
15) Start/Stop Loop
16-22) (Shift+Number) Go to Positions 1 through 7 as defined in the current cycle.
23) (Shift+0) Define home position.
24) Abort current acquisition.
25) Go to Home stage position.
26) Update Motor Position.

Appendix D
Known Bugs and Fixes

1) **Image wrap around when using the rotate, zoom, and offset keys.**

   Problem: The Image in the Focus window may appear wrapped top to bottom when doing these operations. This is not a real effect, and no data is lost in the process, but the image appears distorted, as the top portion becomes the bottom of the image.

   Solution: Stopping/Starting Focusing a few times eliminates the problem.

Index

R

S

T

U

Z