Canadian Light Source
FAR-IR Beamline

The MicroGUI Project, V.0.1

The horizontal microscope's user interface & system integration program.

Author
Jai Willems

Supervisor
Brant Billinghurst

June 25, 2021

Contents

Chapter 1

Introduction

The MicroGUI project is a software development program for the horizontal infrared microscope of the Canadian Light Source's FAR-IR Beamline. This document is designed to present an overview of the program necessary for maintenance and consumer use.

The original horizontal microscope control system consisted of three separate software components: the EDM GUI, Point Grey FlyCapture software, and the THORLABS Kinesis software. From the different operating system requirements, the microscope's control became clunky and awkward. Thus, the prime motivator for the MicroGUI project was the control integration into a sole program allowing an improved and more professional user experience.

In the remaining sections of this document, the program setup/installation will be laid out, the functionality explained, and the program's technical structure and code files will be broken down.

Chapter 2

Program Environment & Installation

The MicroGUI program has a series of dependencies that require installation and configuration before the program can be successfully executed. This chapter will detail the steps to fully set up the necessary program dependencies and initialize the program launcher and the program itself.

2.1 Dependencies

The MicroGUI project requires various dependencies, including a specific Python installation, various package downloads, THORLABS motor stage dependencies, and the FLIR/BlackFly camera dependencies. The following subsections will outline such configuration.

2.1.1 Python Environment

The main coding environment for the MicroGUI project is Python and will need to be installed. For the coding environment to be compatible with the later installed dependencies, a version of Python 3.8.10 is recommended and can be downloaded here.

Once the python environment is set up, we need to install the MicroGUI files. To do this, open up git bash and navigate the directory tree using the `cd` command to the location where the program will reside. Then, clone the MicroGUI repository using the following command:

```
git clone https://github.com/JaiWillems/MicroGUI.git
```

If git bash is not installed on the current computer, then download the repositories ZIP folder[1] folder containing all project files from here. Unzip the downloaded folder to the path of interest.

Next, we need to install the Python dependencies required by the MicroGUI program. To download the dependencies, open the windows command prompt and navigate to the MicroGUI directory where the `requirements.txt` file is located. Run the following command:

[1]Ensure you are cloning from the master branch, this will be the stable version.

```
pip install -r requirements.txt
```

This command will install the appropriate versions of the package dependencies to the current Python environment necessary for the seamless operation of the MicroGUI software.

This concludes the Python environment setup.

2.1.2 THORLABS Dependency Configuration

The horizontal microscope setup contains a THORLABS motor used for mode selection. This section details the steps to configure the THORLABS motor environment.

To begin, download and install the THORLABS APT software found here that corresponds to the system architecture and Python version (Python 3.8.10). This software is necessary for allowing the python modules to interface with the motors firmware.

Lastly, a small configuration of the *thorlabs_ apt* Python library folder (downloaded in section 2.1.1) must occur to ensure proper dependencies are accessible. To accomplish this configuration, navigate the file system to find the *APT.dll* file from the THORLABS APT software. The default installation will result in the *APT.dll* file being found in the following path, `C:/Program Files/Thorlabs/APT/APT Server`. Paste this file in the *thorlabs_ apt* directory located within the Python directory (`...\Python\Python38\Lib\site-packages`).

This concludes the THORLABS motor environment setup.

2.1.3 FLIR Dependency Configuration

To interface with the BlackFly USB3 camera, we will need to download and install two pieces of software: the Spinnaker SDK and the latest Python Spinnaker, both are found here.

Spinnaker SDK

Navigate the FLIR downloads directory into the appropriate operating system and download the `Latest Spinnaker Full SDK` file. Once downloaded, run the executable's installation corresponding to your system architecture (e.g. `SpinnakerSDK_FULL_2.4.0.144_x64.exe`).

Python Spinnaker

Navigate the FLIR downloads directory to the appropriate operating system, open the `Latest Python Spinnaker` folder, then download and unzip the latest Python Spinnaker (PySpin) folder that matches the installed Python version. For Python 3.8 on a Windows 64 machine, the appropriate folder is `spinnaker_python-2.4.0.143-cp38-cp38-win_amd64.zip`).

The PySpin library can then be installed from the Python WHL file located in the just downloaded PySpin directory. To accomplish the download, use the following commands in a terminal to navigate to the PySpin directory and install the WHL file:

```
cd <PySpin_whl_unzip_destination>
pip install --user <PySpin_whl_file>
```

This concludes the FLIR camera environment setup.

2.2 Producing the MicroGUI Launcher

The MicroGUI is initialized using an executable file dubbed the MicroGUI Launcher. This section will detail the steps to create a new launcher executable file if the MicroGUI files' directory tree is altered.

The `run.py` file is simple and only contains three lines of code:

```
import os
os.chdir("C:/Users/Public/Documents/MicroGUI/microgui/source"
os.system("python main.py")
```

For the run file to be properly configured, the path passed in to the `os.chdir()` function must be the path to the MicroGUI's `main.py` file. Once the run file has the appropriate path, navigate the command prompt to MicroGUI's source folder using the `cd` command and use the following command:

```
pyinstaller --noconfirm --onedir --windowed --icon <icon path> --name
 "MicroGUI Launcher" <output path>
```

This will produce the output executable and accompanying files to the `<output path>`. A shortcut can then be made to copy the executable to the desktop.

2.3 Program Operation

At this point, all necessary dependencies are installed, and we are ready to run the program.

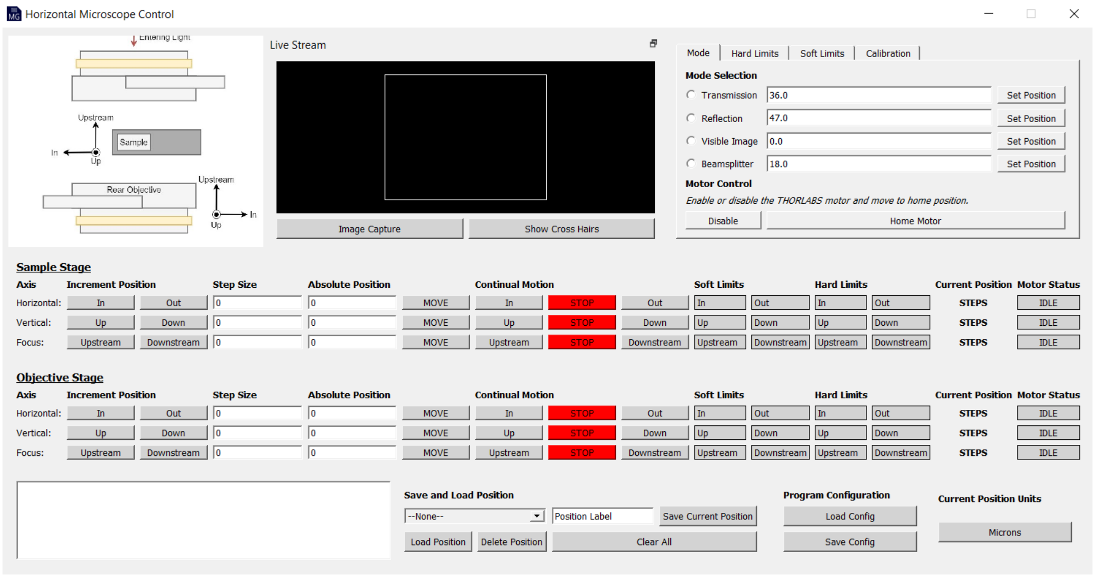
Begin by ensuring **all THORLABS and FLIR/BlackFly software are closed** as their operations have been found to prevent access to external hardware from the MicroGUI software. Then, check that the camera and THORLABS motors are connected to the computer (they will need to be connected before program initiation).

There are two main methods of program initialization: using the launcher or the command line. If there exists a current MicroGUI Launcher, double-click the launcher to activate the program. To initialize the program from the command line, navigate the terminal to the project repository (downloaded in section 2.1.1). Type the command `python main.py` to initialize the program. In both cases, the program will take a moment to start up. Once the GUI has appeared, you will be ready to use the software to interface with the FAR-IR Horizontal Microscope.

Chapter 3

Features

This section details the features of the MicroGUI interface along with their use.



Transmission 36.0
Reflection 47.0
Visible Image 0.0
Beamsplitter 18.0

Figure 3.1: MicroGUI application.

3.1 Diagram Window

The diagram window displays the image in figure 3.2 which defines the coordinate system of the microscope's controls. It defines the horizontal, vertical, and focus axes for the sample and objective stages. It also defines which directions are in and out, upstream and downstream, and up and down, respectively.

3.2 Live Feed Window

The live feed window provides a direct user interface with the FLIR/Point Grey Blackfly USB3 Camera (as seen in figures 3.3 and 3.4). The display has four main components:

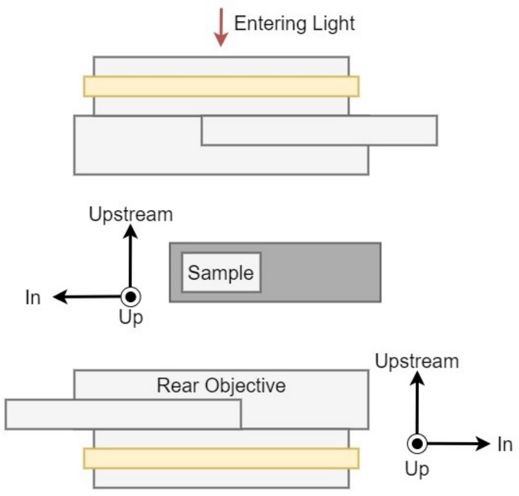


Figure 3.2: MicroGUI movement diagram.

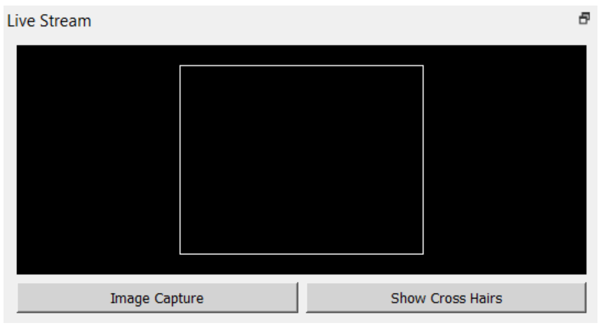


Figure 3.3: Blank live feed, no cross hairs.

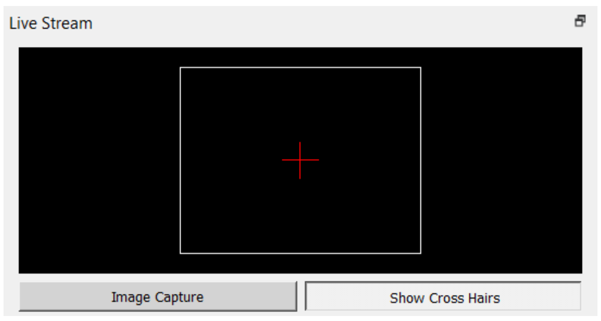


Figure 3.4: Blank live feed, cross hairs.

1. it includes a viewing window that surrounds the camera display to allow users to change the size and position of the live feed. The feed size can be changed by using a mouse scroll wheel while hovering over the feed. The feed can be panned by clicking and dragging the feed.

2. Optional crosshairs allow users to track motion and localize on certain objects. The crosshairs can be toggled on or off using the *Show Cross Hairs* toggle button.

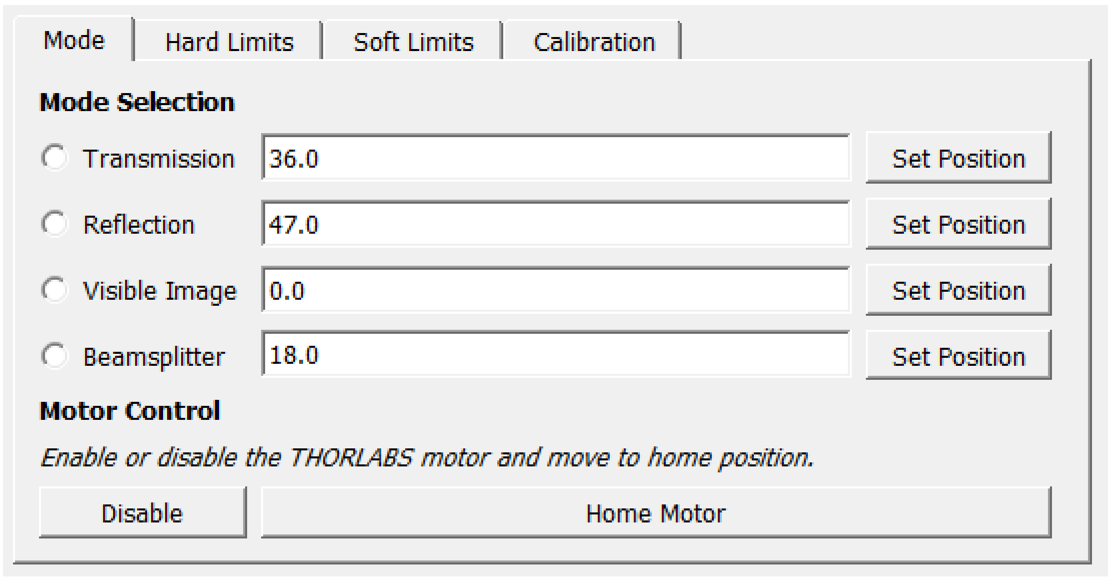
3. An image capture button which, when pressed, will open up a secondary window to allow users to choose the saving directory and filename. Note that the crosshairs will be added to the image if present on the feed display.

4. An enlarged view functionality is available by clicking the *Restore Down* button in the live feeds upper right corner. This will detach the feed and allow users to enlarge the window. To return the live feed to the GUI, double click on the live stream's title bar.

3.3 Tab Window

The tab window contains four tabs detailing less frequently interfaced information yet still essential to access.

3.3.1 Mode Tab



Mode Selection

Transmission 36.0

Reflection 47.0

Visible Image 0.0

Beamsplitter 18.0

Motor Control

Enable or disable the THORLABS motor and move to home position.

Figure 3.5: Mode tab.

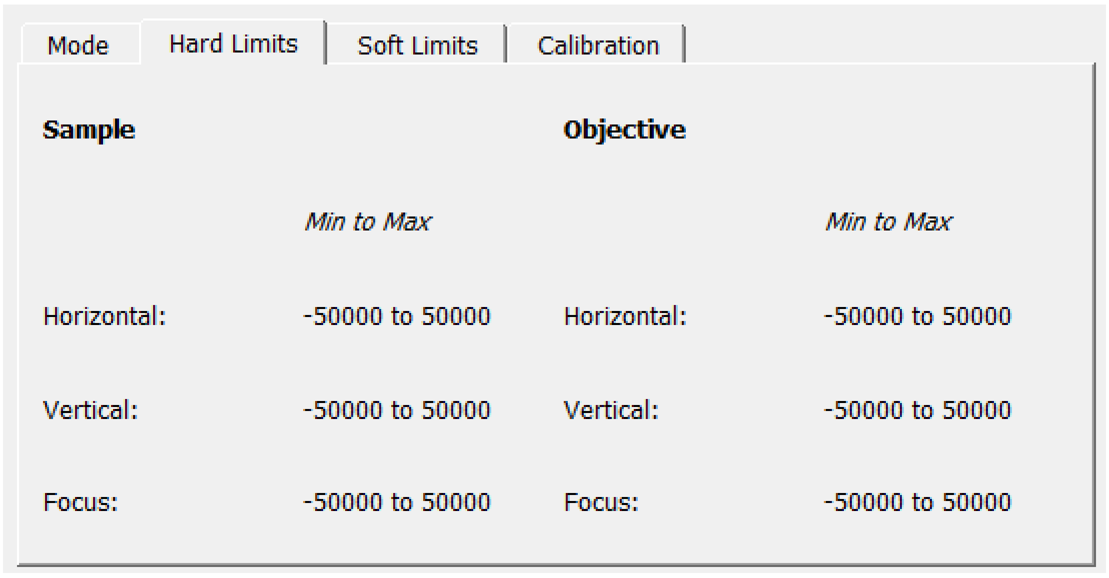
The mode tab, as seen in figure 3.5 provides the necessary controls to change the microscope mode and calibrate motor/mode positions.

There are four mode selection buttons to select between the reflection, transmission, visible image, and beamsplitter modes. The positions associated with each mode will be saved from the previous program execution but can be altered using the adjacent line edits. To change a mode position, change the modes adjacent line edit (in mm) before selecting the *Set Position* button to save the inputted position value to the internal database. If the mode position being changed is selected, the motor will automatically adjust to the newly set position.

The bottom of the mode tab has additional motor controls, which will be irrelevant to typical use. These buttons allow the user to enable or disable the mode motor as well as move the motor to its *home* position. Note that the motor must be enabled to change modes or home successfully.

3.3.2 Hard Limits Tab

The hard limits tab, as seen in figure 3.6 details the range of motion for each of the sample/objective motors. These limits are defined by the physical constraints that illuminate the hard limits and are displayed primarily for the scale of motion and user insight. Note that the hard limit values are given in steps.



Sample	Min to Max	Objective	Min to Max
Horizontal:	-50000 to 50000	Horizontal:	-50000 to 50000
Vertical:	-50000 to 50000	Vertical:	-50000 to 50000
Focus:	-50000 to 50000	Focus:	-50000 to 50000

Figure 3.6: Hard limits tab.

3.3.3 Soft Limits Tab

The soft limits tab, as seen in figure 3.7 allows the user to set soft limits on the program to prevent possible hardware collisions. Such limits can be set in three ways:

1. The *Set Soft Limits* button will set the soft limits to the values inputted into the 12 text boxes. If an inputted soft limit value is more extreme than its corresponding hard limit value, then the soft limit will be set to the hard limit. All motors that fall outside of the new limits will automatically be moved to the closest limit. Additionally, the minimum soft limit must be less than the maximum soft limit for the limit to be set.

2. The *Set Minimum Soft Limits* button will set all soft limits to zero. All motors that fall outside of the new limits will automatically be moved to the zero position.

3. The *Set Maximum Soft Limits* button will set all soft limits to the corresponding hard limit values (indicated in the hard limits tab) to effectively remove any soft limit constraints. This method is not recommended as the soft limits provide a safety margin against hardware collisions. Only use this functionality if on-site and in the visual range of the microscope setup.

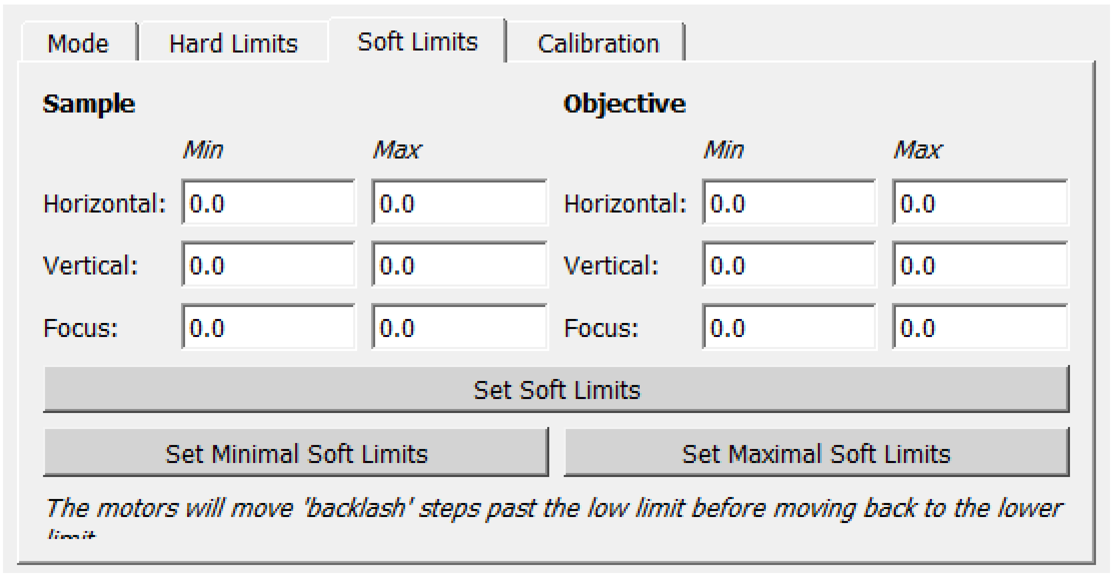
Note that soft limits are interfaced in units of steps.

3.3.4 Calibration Tab

The calibration tab seen in figure 3.8 allows users to manipulate the data display and motor parameters for best performance. This tab contains several controls:

1. The *ZERO* button functionality allows users to zero the position of a stage motor to create a new datum. Zeroing will change the soft limits, hard limits, and current position to reflect the new zeroed position. Subsequent zeroing can only be done when displaying relative values.

2. Backlash line edits and the *Set Backlash Values* button allow users to define and



Sample

	Min	*Max*
Horizontal:	0.0	0.0
Vertical:	0.0	0.0
Focus:	0.0	0.0

Objective

	Min	*Max*
Horizontal:	0.0	0.0
Vertical:	0.0	0.0
Focus:	0.0	0.0

The motors will move 'backlash' steps past the low limit before moving back to the lower limit

Figure 3.7: Soft limits tab.

save new backlash values for each stage motor. Inputted values represent steps and must be positive integer values. It is recommended that users do not change default values that have been proven through testing to be optimal.

3. The *Display Actual Values* button toggles the displayed values to show either actual or relative values. When pressed, all values displayed will represent the actual positions of the limits and motors concerning an internally defined reference position. When unpressed, all values displayed will represent the positions of the limits and stage motors with reference to the user-defined datum (a result of zeroing a motor's position). In other words, unpressed allows the users to see positions in the zeroed domain.

4. The *GLOBALS* button provides insight into the program's internal variables for debugging and validation purposes.

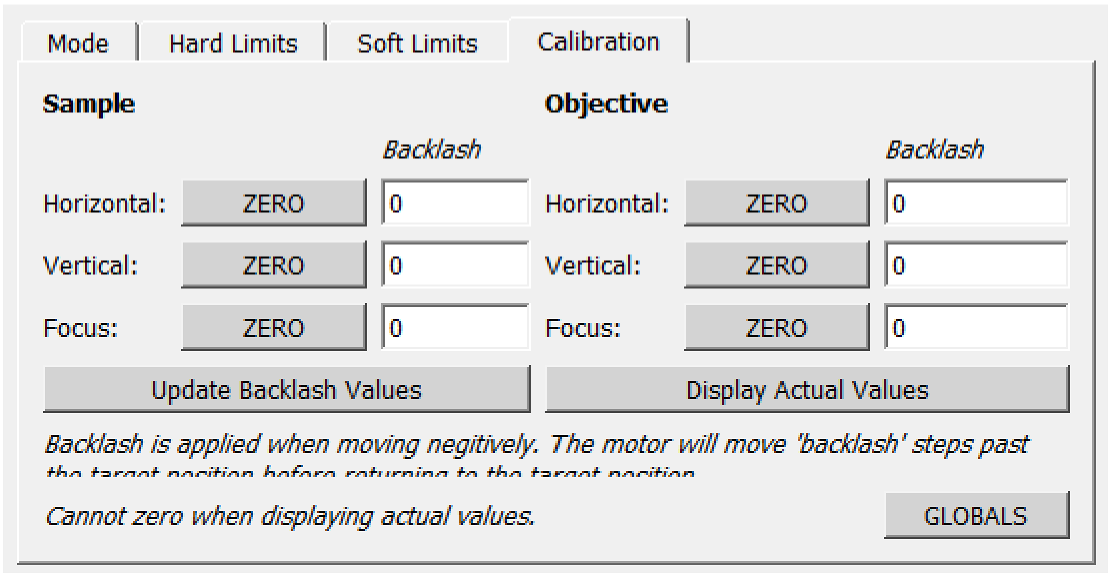
3.4 Sample & Objective Windows

The sample and objective windows have similar controls and nearly identical functionality. As a result, the information in this section can be generalized to both displays.

The displays can be broken into four main sections: three-movement forms (incremental, absolute, and continuous positioning) and one for motor status.

1. The sample and objective stage motors can be moved with precision using the increment-position controls. These controls include incremental positive and negative buttons in addition to a step-size input. The step size indicates the size of each increment in steps, and the incremental position buttons indicate the direction to move the stage. These methods are ideal for precise motor control.

2. The sample and objective stage motors can also be moved to an absolute position using the absolute position controls. An absolute position can be inserted into the *Absolute Position* line edit, and the *MOVE* button can be pressed to start motor

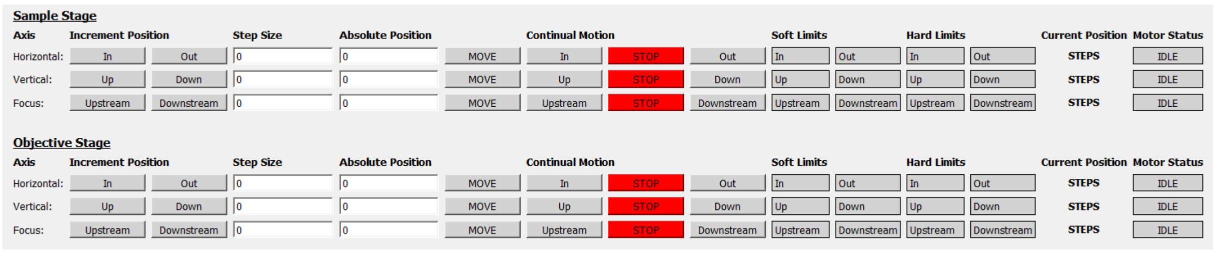


Sample			**Objective**		
		Backlash			*Backlash*
Horizontal:	ZERO	0	Horizontal:	ZERO	0
Vertical:	ZERO	0	Vertical:	ZERO	0
Focus:	ZERO	0	Focus:	ZERO	0

Backlash is applied when moving negitively. The motor will move 'backlash' steps past the target position before returning to the target position

Cannot zero when displaying actual values.

Figure 3.8: Calibration tab.



Sample Stage

Axis	Increment Position		Step Size	Absolute Position		Continual Motion			Soft Limits		Hard Limits		Current Position	Motor Status
Horizontal:	In	Out	0	0	MOVE	In	STOP	Out	In	Out	In	Out	STEPS	IDLE
Vertical:	Up	Down	0	0	MOVE	Up	STOP	Down	Up	Down	Up	Down	STEPS	IDLE
Focus:	Upstream	Downstream	0	0	MOVE	Upstream	STOP	Downstream	Upstream	Downstream	Upstream	Downstream	STEPS	IDLE

Objective Stage

Axis	Increment Position		Step Size	Absolute Position		Continual Motion			Soft Limits		Hard Limits		Current Position	Motor Status
Horizontal:	In	Out	0	0	MOVE	In	STOP	Out	In	Out	In	Out	STEPS	IDLE
Vertical:	Up	Down	0	0	MOVE	Up	STOP	Down	Up	Down	Up	Down	STEPS	IDLE
Focus:	Upstream	Downstream	0	0	MOVE	Upstream	STOP	Downstream	Upstream	Downstream	Upstream	Downstream	STEPS	IDLE

Figure 3.9: Sample and objective control windows.

motion. This control sequence is recommended when using predefined positions. Note that a newly inputted value is not internally known to the program until the move button is pressed. Inputting a value and not pressing the move button will cause nonalignment between the *Current Position* and *Absolute Position* displays; it will also influence the zeroing and incremental functionality.

3. The final type of motion is continuous. The continuous motion controls include the continuous positive and negative buttons and a *STOP* button. The pressing of the continuous positive or negative push buttons will begin the stage motor's motion in the defined direction. Such motion will continue until a soft/hard limit is reached, or the *STOP* button is pressed. This form of control is ideal when large and coarse movements are desirable.

4. The sample and objective stages also encapsulate several status indicators, including soft and hard limits, current position feedback, and motor status labels.

   (a) The soft or hard limit labels will illuminate green if their respective soft or hard limits are met. When a specific limit is not met, it will default to light grey.

   (b) The *Current Position* feedback display updates continuously to give the most up-to-date position of the motor. The current position display and absolute position line edit should align under most circumstances, but the current position label should be taken as true if these displays have diverged.

(c) The motor status label indicates the current state of the motor. The possible states include idle, powering, powered, releasing, active, applying, and unpowering.

3.5 Miscellaneous Window

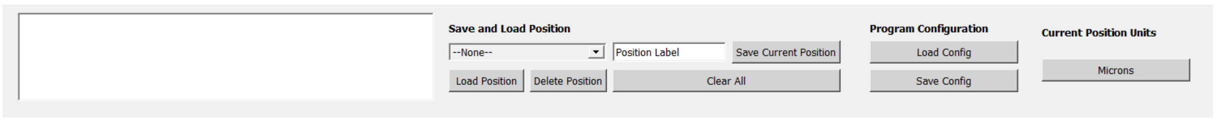


Figure 3.10: Miscellaneous window.

The bottom of the GUI represents the miscellaneous tab with categorized functionality consisting of four main sections:

1. The *Program Status Console* is the largest widget on the miscellaneous window, which prints out statements throughout the program's operation. These messages provide confirmation of user actions and provide warnings and errors should they arise. For this program, *warnings* are defined as changes the GUI has made to a user action to adhere to internal rules that do not inhibit the program but may change the expected outcome. Warnings will be displayed in yellow text. An *error* is defined as inhibited GUI functionality due to command incompatibility with the software/hardware. Errors will be displayed in red text.
2. The *Save and Load Position* section allows the user to store and restore positions of interest. There are four main parts of the *Save and Load Position* section.
   (a) A user can save the current sample and objective positions by adding in a position label and pressing the *Save Current Position* button. Note that position labels must be unique and will throw an error if a duplicate label is inserted.
   (b) A user can load a previously saved position by selecting the desired position from the position combo box and pressing the *Load Position* button.
   (c) A user can delete a previously saved position by selecting the desired position from the position combo box and pressing the *Delete Position* button.
   (d) Lastly, a user can clear all saved positions by pressing the *Clear All* button.
3. The miscellaneous tab also contains *Program Configuration* controls. The *Load Config* button allows a user to upload a previous program configuration, whereas the *Save Config* button allows a user to save the current configuration.
4. Lastly, the *Microns* button toggles the sample and objective *Current Position* displays between units of steps and micrometres.

Chapter 4

MicroGUI Code

4.1 File Structure

The MicroGUI project is separated into eight source code files that combine to give the user the functionality they desire.

- The `main.py` file initiates and closes the program by invoking all necessary files to work in parallel to create the user interface and underlying functionality.
- The `gui.py` file is responsible for creating the interface by placing various widgets and windows.
- The `controller.py` file adds the functionality to the user interface by connecting the interactive widgets created by `gui.py` to control sequences to invoke change in the microscope or display.
- The `configuration.py` file provides the functionality for loading and saving the program configuration files and the saved positions configuration file.
- The `config.json` file stores the macro variables from the most recent program execution and uploads them upon subsequent program instantiation to ensure program variables are saved.
- The `saved_positions.json` file stores the programs saved positions.
- The `thorlabs_motor_control.py` file branches the gap between `controller.py` and the THORLABS API.
- The `flir_camera_control.py` file bridges the gap between `controller.py` and the cameras Spinnaker SDK.

4.2 Files

4.2.1 main.py

The `main.py` file calls the proper code sequence to execute all necessary files.

Documentation Resources

None

Code Overview

Refer to table 4.1 for detailed code overview.

Line(s)	Description	Notes
8 - 18	Module imports.	
22	Initialize the THORLABS motor.	Initialization enables the motor, and moves to the homing position.
26 - 27	Import the default configuration file and saved positions configuration file. save all data to program variables.	
30 - 46	A function called on program closure to exit the application and stop all stage motors.	
50 - 55	Create and show the GUI.	

Table 4.1: main.py code overview.

4.2.2 gui.py

The `gui.py` file creates the user interface display.

Documentation Resources

- PyQtGraph: `https://pyqtgraph.readthedocs.io/en/latest/`
- PyQt5: `https://doc.bccnsoft.com/docs/PyQt5/`
- Qt for Python: `https://doc.qt.io/qtforpython/`

Code Overview

Refer to table 4.2 for detailed code overview.

Line(s)	Description	Notes
9 - 43	Module imports	
272 - 303	GUI class initialization code.	The initialization code sets the main GUI layout and calls various methods to create key GUI displays.
305 - 318	The diagram window initialization which imports and displays the microscope diagram.	

331 - 536	Sample window configuration.	Lines 339 - 355 define the main window and the column labels. Lines 362 - 536 are broken into three sections, one for each of the sample dimensions. Each dimension's code is further broken down into three sections: (1) widget initialization, (2) Widget styling, and (3) adding the widgets to the window layout.
538 - 743	Objective window configuration.	Follows an identical format to the sample window configuration.
745 - 825	Create the base window display.	Lines 754 - 757 initialize the text browser window. Lines 760 - 785 initialize the save and load position functionality. Lines 788 - 801 initializes the program configuration options. Lines 804 - 823 initialize the unit conversion functionality.
828 - 953	Class to produce the detachable camera live feed.	Lines 852 - 862 set the dockable fucntionality and connect buttons to control sequences. Lines 872 - 903 are run to update the current image displayed on the live feed display. Lines 887 - 893 alter certain pixel colours to create the figure's cross hairs. Lines 905 - 918 initialize the camera display. Lines 920 - 932 Organize the window and define the *Image Capture* and *Show Cross Hairs* buttons. Lines 938 - 953 create the image capture functionality.
957 - 1397	The class used to initialize the table window.	Lines 1084 - 1105 define the main window and layout. Lines 1112 - 1169 detail the mode selection and configuration tab. Lines 1176 - 1221 detail the hard limits tab. Lines 1228 - 1303 detail the soft limits tab. Lastly, lines 1310 - 1397 detail the calibration tab.

Table 4.2: gui.py code overview.

4.2.3 controller.py

The `controller.py` file adds the functionality to the user interface.

Documentation Resources

- PyQt5: https://doc.bccnsoft.com/docs/PyQt5/
- Qt for Python: https://doc.qt.io/qtforpython/
- PyEpics: https://pyepics.github.io/pyepics/overview.html

Code Overview

Refer to table 4.3 for detailed code overview.

Line(s)	Description	Notes
9 - 39	Module imports.	
43	Initialize EPICS environment.	The line is necessary to connect to the CLS EPICS network.
206 - 389	Initialize the GUI.	The GUI is initialized by setting the display values to those of the macro variables and setting PV values.
391 - 523	Connect GUI widgets to a control sequences.	
525 - 551	Change the microscope mode when one of the mode radio buttons are selected.	
553 - 591	Control sequence to update the macro mode positions to the inputted values.	
593 - 604	Control sequence to enable or disable the THORLABS motor.	
606 - 621	Control sequence to home the THORLABS motor.	
623 - 673	Control sequence to increment position.	If the increment takes the position past a limit, it will move the motor to the limit.
675 - 721	Control sequence for moving to an absolute position.	If the inputted position is out of the limits, the motor will move to the limit.
723 - 755	Control sequence for continuous motion.	The motor will stop at the limit if the stop button is not pressed.
757 - 786	Callback function that is run when the absolute position PV value changes. This function updates the absolute position display.	
788 - 817	Provides an offset which can be added or subtracted from an absolute or relative position to convert to the other.	Inversion increases the functionality of this method.

819 - 950	Update soft limit macro variables.	The macros are updated, the soft limit displays are updated for consistent formatting, the indicators are set if the limits become less then the motor position, and motors are moved if found outside of the new limits.
952 - 985	Control sequence to update the backlash PV's.	
987 - 1027	Control sequence to zero a stage motor.	The macro relative and absolute position variables are updated and the display values are updated.
1029 - 1102	Callback function that runs to update the motor status display when the motor state changes.	Soft limit indicators will be checked whenever the motor is not active or powering/unpowering.
1104 - 1131	Control sequence that checks if any motors are outside of their limits and will move those found outside of their limits back within them.	
1133 - 1173	Callback function to set the hard limits if the associated PV has a change in value.	
1175 - 1225	Control sequence to check and set the soft limit indicators.	
1227 - 1297	Control sequence to change the display values between actual and relative values.	The sequence changes the soft and hard limits, absolute position display, and current position display.
1299 - 1335	Callback function to update the current position label.	
1337 - 1254	Append text functionality to write to the text browser widget.	
1356 - 1365	Print global variables for debugging and program insight.	
1367 - 1389	Methods to load and save configuration files.	
1391 - 1413	Method to changed the displayed units for the current display.	
1415 - 1462	Methods for the save, load, delete, and clear positions functionality	

Table 4.3: controller.py code overview.

4.2.4 configuration.py

The `configuration.py` file encapsulates functionality used for the saving and loading of configuration files.

Documentation Resources

None

Code Overview

Refer to table 4.4 for a detailed code overview.

Line(s)	Description	Notes
8 - 9	Module imports.	
12 - 33	Load configuration functionality which opens the configuration file and returns the files information in two formats.	The files information is formatted to be able to pass into the save configuration functionality.
36 - 51	Save configuration functionality which saves the information passed in to a configuration file defined by the *path* parameter.	
54 - 93	These two function deal with the transformation of information between nested and linear dictionaries necessary for the load and save configuration functionality.	For the `condense_macros()` function, the keys in `macros` must be present in `baseDict`.
96 - 114	Load saved position functionality uploads previously saved positions to the program.	
117 - 131	Save positions functionality which writes all saved positions to the saved positions configuration file.	

Table 4.4: configuration.py code overview.

4.2.5 config.json

The `config.json` file defines macro variables used across scripts.

Documentation Resources

None

Code Overview

Refer to table 4.5 for detailed code overview.

Line(s)	Description	Notes
2 - 6	Define the THORLABS mode positions.	
8 - 23	Defines the base and relative positions.	The base and relative positions are used for the zeroing functionality. The base position relates the zeroed domain to the actual domain and the relative position is the current position in the zeroed domain.
26 - 32	Defines the conversions between microns and steps.	The conversions factors are given by *microns/step*.
34 - 61	Defines soft and hard limit values.	
64 - 70	Defines the backlash values.	
72 - 179	PV names for all motor stages.	The naming convention follows the format "axisobjecttext". The axis can be "X", "Y", or "Z" corresponding to the horizontal, vertical, and focus, respectively. The object can be "S" or "O" for the sample or objective stages respectively. Finally, the text describes the PV use.

Table 4.5: config.json code overview.

4.2.6 saved_positions.json

The `saved_positions.json` file saves positions saved during program runtime.

Documentation Resources

None

Code Overview

The `saved_positions.py` file will have varying data due to different saved configurations from run time.

4.2.7 thorlabs_motor_control.py

The `thorlabs_motor_control.py` file interfaces with the THORLABS motor to configure the microscope mode. The file configures the motor properties, state and provides position control.

Documentation Resources

- thorlabs_apt: https://github.com/qpit/thorlabs_apt

Code Overview

Refer to table 4.6 for detailed code overview.

Line(s)	Description	Notes
8	Module imports.	
11 - 35	Initialize the motor.	Configure other motor parameters by inserting code between lines 33 and 35. Other parameters are detailed in the thorlabs_apt source code. An exception will be raised if not motor is connected.
38 - 59	Enable and disable control over the motor stage.	
62 - 70	Home motor functionality.	
73 - 102	Changes the current position of the motor.	

Table 4.6: thorlabs_motor_control.py code overview.

4.2.8 flir_camera_control.py

The `flir_camera_control.py` file interfaces with the FLIR camera to set up camera properties and capture images.

Documentation Resources

- simple-pyspin: https://klecknerlab.github.io/simple_pyspin/

Code Overview

Refer to table 4.7 for detailed code explanation.

Line(s)	Description	Notes
8 - 9	Module imports.	
12 - 31	Image acquisition function.	Insert camera configuration code between lines 23 and 27. Lines 27 - 29 contain the image acquisition functionality.

Table 4.7: flir_camera_control.py code overview.