

# FTIR Spectrum

## Users' Guide

### Introduction and Installation

This program was developed within Anaconda 5.1.0, which contains all of the prerequisite libraries (except possibly pyqtgraph.) Development completed in August 2018.

Questions may be directed to Jiayi Tang, [jiayi@alumni.ubc.ca](mailto:jiayi@alumni.ubc.ca).

Nonstandard libraries used are:

- numpy
- pyqtgraph
- pandas
- scipy

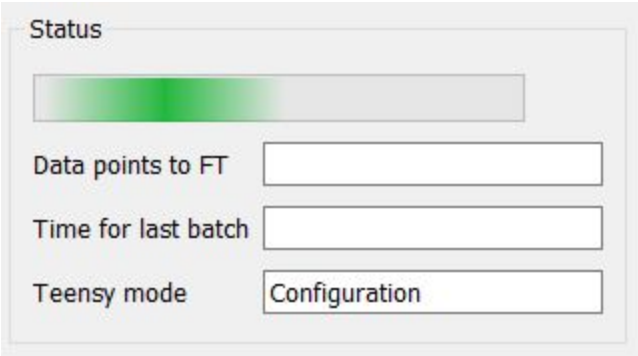
### Teensy

The Teensy has two modes: measurement and configuration. Switching between the two modes is performed via the push button.

- The reference intensity signal should be plugged into pin A0 after it is scaled to be between 0-3.3V. Taking advantage of this full range is ideal (ie. the signal should be scaled to fill up 0-3.3V.)
- The measurement intensity signal should be plugged into pin A19 after scaling.
- The digital trigger for the stage changing directions should be plugged into digital pin 11.

### Status box

In configuration mode, the box displays the current mode. It may also display information left in the boxes from before configuration mode was activated.



The screenshot shows a window titled "Status". Inside the window, there is a horizontal progress bar that is mostly green with a small grey section on the right. Below the progress bar, there are three rows of labels and input fields:

Data points to FT	<input type="text"/>
Time for last batch	<input type="text"/>
Teensy mode	<input type="text" value="Configuration"/>

In measurement mode, the box displays the current mode, as well as how many points are passed to the Fourier transform for processing. This will be less than 100% because of clipping to make the signal symmetric, as well as an additional clipping of the ends specified by the user.

This percentage can be seen as an “efficiency” for how much scanning distance is wasted due to clipping for symmetry. The efficiency is improved when the stage is set so that its motion is centered about the zero-path-difference (ZPD) point for the two mirrors.

The box also displays the time it took for the last batch of data to arrive. This is generally controllable by specifying the scan distance and speed, with some overhead due to processing. For longer periods of time (this threshold time can be configured via config.ini > GUI > progress\_bar\_min\_time) the progress bar will display how many data points have been collected towards the next batch of data.

The Status window contains a green progress bar at the top. Below it, there are three rows of data:

Data points to FT	11908/15248 (78.1%)
Time for last batch	0.56 s
Teensy mode	Measurement

## Plotting Controls

The Plotting controls window includes the following elements:

- ☐ Freeze plot
- ☒ Display intensity/position
- ☐ CW mode
- ☐ Logarithmic plot
- Tukey window with alpha: 0.10 (with a dropdown arrow)
- % to clip data: 0.10 (with a dropdown arrow)

UI element	Function
Freeze plot checkbox	When checked, prevents plots from being updated, and holds the current displayed data onto the screen. Data is continuously acquired in the background, so when the box is unchecked, the plots will update to the most recent available data.
CW mode	When checked, disables clipping for symmetry about ZPD point (since there is none for continuous wave lasers.) Changes the “Data points to FT” box in the Status group to instead display a ratio of zero-crossings between the reference and measurement signals. Plots the reference laser intensity signal instead of the

	positions that are computed based on that data.
Display intensity/position	<p>When checked, displays a plot of measurement laser intensity and position against index (ie. the position in the data queues acquired from the Teensy.) Index 0 is set to be the zero-path-difference location when not in CW mode.</p> <p>Uncheck to display only the spectrum plot, which will expand to fill more screen space.</p>
Logarithmic plot	<p>Check to plot the spectrum with the y-axis as a log scale. The calculations may cause lag in the program, but as the x-range of the plot becomes smaller, the program will have to compute less values and become more responsive. Initially zooming in the x-axis may be not very responsive--preservering has generally worked. If it is truly unworkable, scan for a shorter distance to reduce the spectral resolution, switch to log mode, zoom into the desired x-range, then increase the scan distance again.</p>
Drop down menu with windowing functions	<p>Choose a windowing function to apply to the measurement intensity data. Currently, the choices are:</p> <ul style="list-style-type: none"> <li>• No window (no tapering at ends of data; observed to introduce noise at high wavelengths in spectrum)</li> <li>• Gaussian window <ul style="list-style-type: none"> <li>○ The parameter input box right of the drop down box becomes available to input a standard deviation (SD) of the window. A higher SD will make the window broader and make the spectrum focus less on the mode-locked component of the spectra, if there is a blend of CW and mode-locked signal.</li> </ul> </li> <li>• Tukey window <ul style="list-style-type: none"> <li>○ The parameter input box is used to specify alpha, which is the percent of data which will be affected by the Tukey window's taper. The Tukey window leaves the remainder of the data in the center unaffected. For example, if alpha is given to be 0.1, the first 5% of the data will taper from 0 to full intensity, the middle 90% will be unaffected, and the last 5% will taper from full intensity to 0. The shape of the taper is made with stretched cosines.</li> </ul> </li> </ul> <p>Acceptable ranges for these parameters, their defaults, and the step size of the entry boxes (when the up and down arrows in each box are pressed) can be configured in config.ini under DataProcessing.</p>
% to clip data	<p>Intensity and position data will be clipped by half of this percentage at the beginning and end. This is intended to reduce the impact of irregular behaviors when the stage changes directions. This</p>

	clipping happens before the array is clipped again for symmetry about the ZPD point. Acceptable ranges, default, and step size for the margin can be configured in config.ini under DataProcessing.
--	--

## Data collection

Data collection

Filename formatting

☐ Save raw data      ☐ Save spectrum

This box controls what data to save and how the produced filenames are formatted.

The text box contains formatting which is passed to the strftime method in Python’s datetime. See <https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior>. As an example, we can add “HeNe” in front of “%Y%b...” to get filenames that are saved like “FringesGHeNe2018Jul ... .csv”.

Filenames are formatted like this:

Fringes Spectrum	N G T	2018Jul20_161819	.csv
Indicates if raw data or spectrum is saved in this file	Prefix of windowing function used (N for none, G for Gaussian, T for Tukey)	Timestamp; customizable by user through text box. Can include arbitrary substrings to add information for which laser was used, etc.	File extension

The two checkboxes save the raw data or the spectrum. If Freeze plot is not on and the Teensy is in measurement mode (ie. collecting data towards the next batch,) the program waits for the next batch and saves that. If Freeze plot is on, the program saves what is currently being displayed. Even if Display intensity/position is off, the fringes can still be saved.

## Stage controls

Stage controls

Start

Starting pos (mm)

6.50

Stop

Distance to scan (mm)

0.50

Stage speed (mm/s)

1.00

Starting stage:  
'k1\n>\$ \n\r'  
Data saved to FringesT2018Jul27\_170236.csv  
Stopping stage:  
'k0\n>\$ \n\r'

Read settings

Set center

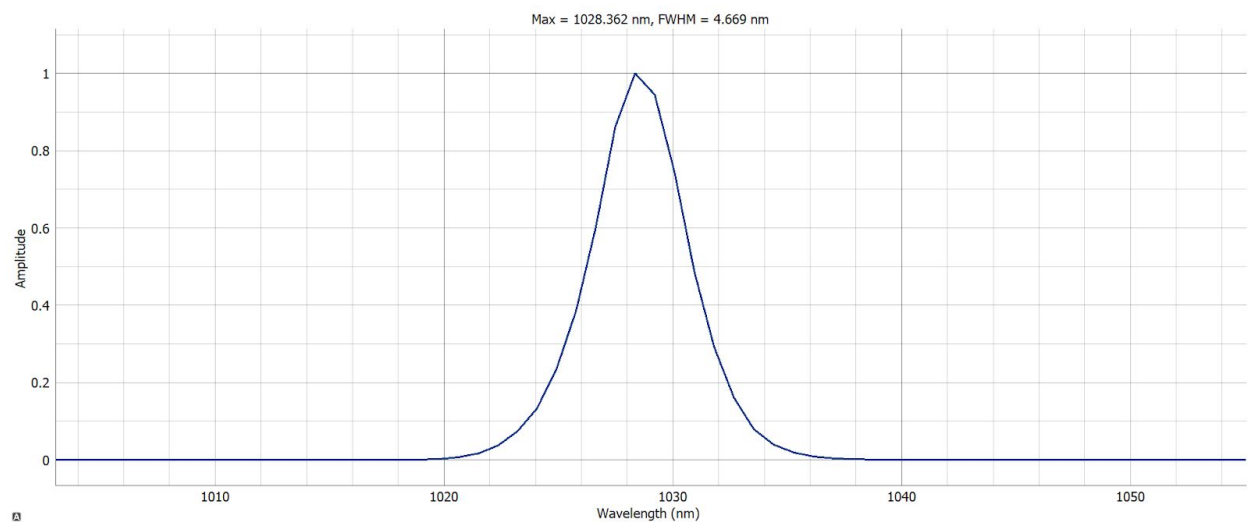
Scan stage

Send

UI element	Function
Start button	Starts the stage. The stage should retain its previous settings.
Stop	Stops the stage.
Starting pos (mm)	Specify in mm how far from one end of the stage it should begin its motion.
Distance to scan (mm)	Specify how far the stage should move before turning back and moving the same amount in the opposite direction.
Stage speed	Specify rate at which stage should move. Too high of a scan speed means Teensy cannot read values from detector fast enough, and data may be lost due to aliasing of signal. Too low of a scan speed may result in unreliable results, since the stage's motion is controlled via a servo and prone to jerkiness back and forth at low speeds.
Read settings	Sends commands to the stage controller to query what the currently programmed settings on the stage are. These are translated into physical parameters and the values in Starting pos, Distance to scan, and Stage speed are changed accordingly (ie. the current values in these boxes is overwritten.)
Set center	Commands stage to move to the position given in Starting pos. The stage stays there and does not scan.
Scan stage	The values in Starting pos, Distance to scan, and Stage speed are commanded to the stage controller and the stage begins to move according to these specifications. Note that these are approximate due to a conversion from physical measurements to discrete digital levels in the stage controller. Increased accuracy of actual distance travelled and speed can be acquired from counting reference laser

	fringes and by using the time duration display in the Status box.
Console	<p>Displays replies from the stage controller after commands are sent. If commands are sent and read properly, the commands should be mirrored back. Also contains information if the program catches exceptions. Rarely, a batch of data may be badly behaved or otherwise abnormal. This would normally cause the program to throw an exception, but this is caught as the most vulnerable parts of the program are wrapped in Python's try-except blocks. If the console informs that errors are persistently occurring, the actual console (black and white text-based window) will contain error messages and tracebacks for more detail.</p> <p>Allows custom commands to be sent to the Teensy via the text box. Commands should be made in accordance with the syntax of the stage controller (SCA814.) Commands can be sent via pressing Enter or by pressing the Send button.</p>

## Plots



See also [http://pyqtgraph.org/documentation/mouse\\_interaction.htm](http://pyqtgraph.org/documentation/mouse_interaction.htm) for additional interactions.

Mouse interaction	Function
Left click + pan on plot area (gridded area above)	Pans plot without constraint. On intensity/position plot, this will pan both intensity and position horizontally, but only intensity will pan vertically.
Left click + pan on axis	Pans plot, with constraint to that axis.

	On intensity/position plot, the intensity plot is controlled with the left axis, and the position plot with the right axis.
Scroll on plot area	Zoom in/out of plot, along x and y.
Scroll on axis	Zoom in/out of plot, with constraint to that axis.
Small A in lower left corner	Reset plots to default x, y ranges. This fits the available data so that it is all displayed.

## Configuration file

This documentation refers to “config.ini”, which is a configuration file found in the same folder as the Python file “FTIR\_DualLaser4.py”. It can be edited with Notepad or another plain text editor. This configuration file holds several options which can be changed by the end-user. Note that if the options are set to something which does not physically make sense (ex. Setting range for margin percentage outside of [0,1]) errors may occur.

Changes made via config.ini will apply the next time the program starts. Generally, things affected by the configuration file can be identified by their UPPERCASE\_STYLE within the code.

[ GUI ]	
spectrum_plot_color	RGB tuple for color of line used to plot spectrum
intensity_plot_color	RGB tuple for color of line used to plot measurement laser intensity
position_plot_color	RGB tuple for color of line used to plot position
progress_bar_min_time	Threshold time for when progress bar should display progress. If batches take less than this time, the progress bar will display a strobing busy indicator instead of rapidly flashing as batches complete.
ms_before_reading_settings	Time to wait between calls of sendReadInstruction and readStageSetting.
spectrum_plot_font_size	Font size to use for spectrum plot.
spectrum_default_xmin	xMin to use on wavelength plot when program initializes.
spectrum_default_xmax	xMax to use on wavelength plot when program initializes.

spectrum_plot_line_width	Thickness of line to use in plotting spectrum.
[DataAquisition]	
progress_bar_update_interval	Interval at which InteractWithTeensy should report its current batch size to GUI thread. If this is lower, the progress bar will fill more smoothly, but there are diminishing returns on the “smoothness” of motion.
secs_btw_serial_retry	If the serial connection to Teensy cannot be initialized at the beginning of the program, wait this many seconds before retrying to open the connection. There are indefinitely many retries, separated by this time interval.
serial_port	The serial port to be used for communication with Teensy.
serial_timeout	How much time to wait for Teensy to output 64 bytes (which it would do in Measurement mode) before deciding that the Teensy is in Configuration mode.
stage_min_step_size	Minimum step size in counts for the stage. Smaller steps may result in finer motion, but more noise.
stage_pause_microseconds	How many microseconds to pause the stage at the ends of its motion. May help with stability.
stage_default_start_pos	Default starting position of stage, written into the corresponding box when the program initializes.
stage_default_scan_dist	Default scan distance of stage, written into the corresponding box when the program initializes.
stage_default_speed	Default speed of stage, written into the corresponding box when the program initializes.
[DataProcessing]	
center_finding_partition_width	Partition size to use when trying to determine center of intensity signal. See programmers’ reference for more information on how this is used. Generally, a smaller partition size results in worse performance, but possibly more accurate center-finding.



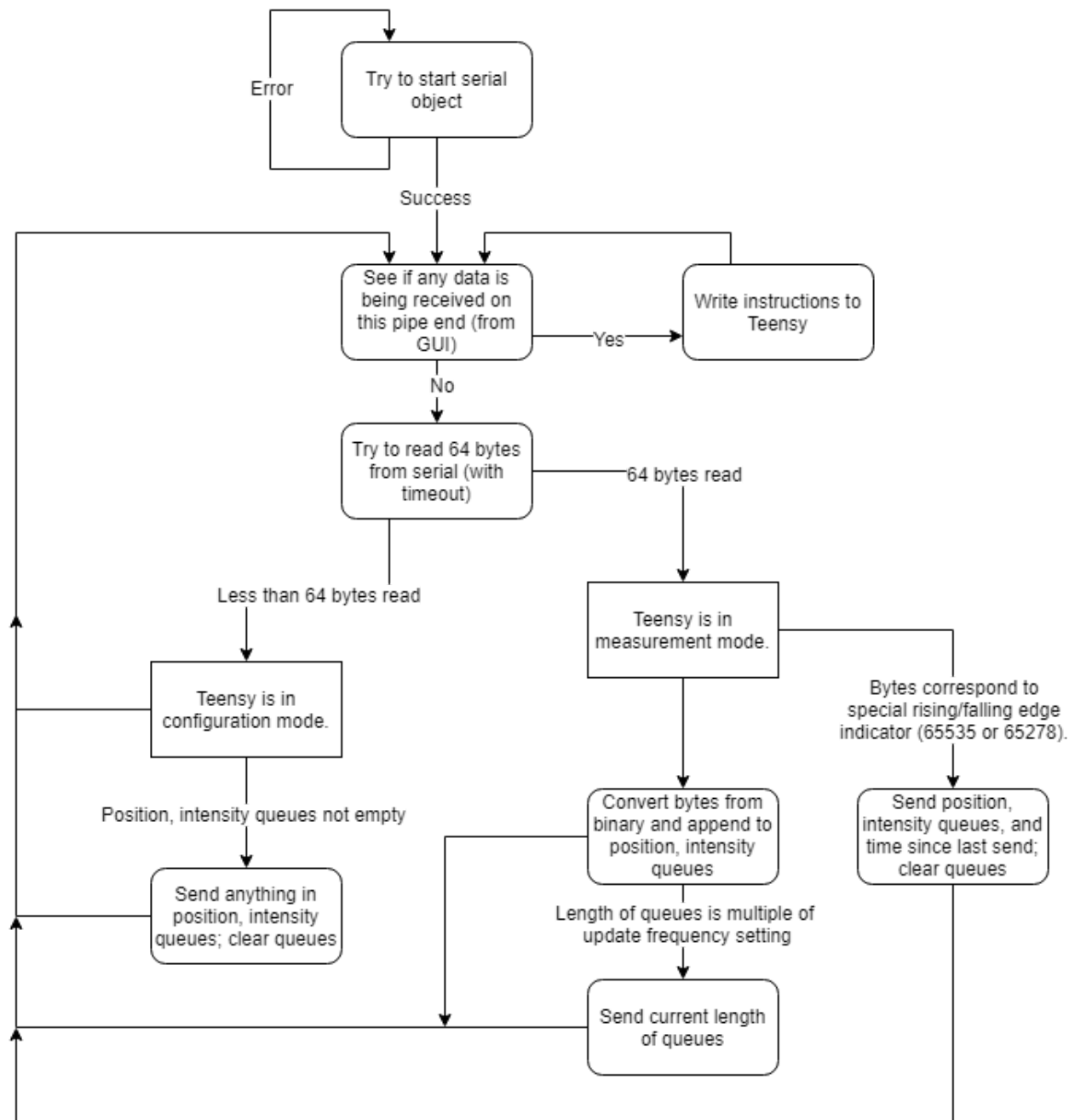
moving_avg_batch_size	How many samples each moving average will consist of. Higher means the smoothing will be more aggressive (possibly to the point of smoothing out fringes.)
gaussian_kernel_stdev	Standard deviation to pass to Gaussian kernel smoothing. Higher means more aggressive smoothing, possibly with loss of meaningful data, as well as worse performance.
gauss_window_min_stdev	See documentation under Plotting Controls.
gauss_window_max_stdev	
gauss_window_def_stdev	
gauss_window_stdev_step	
tukey_window_min_alpha	
tukey_window_max_alpha	
tukey_window_def_alpha	
tukey_window_alpha_step	
margin_min	
margin_max	
margin_default_pct	
margin_step	
[PhysicalConstants]	
speed_of_light	Speed of light. By default this is the accepted value, divided by the refractive index of air.
ref_laser_wavelength	Wavelength of laser used for reference. A HeNe has 632.8nm wavelength.
stage_mm_per_count	How many millimeters one stage count corresponds to. For the VCS10, there are $2^{16}$ counts for 10mm of motion.
stage_tick_microseconds	How many microseconds comprise one tick of the stage controller. For the SCA814, this is 25.6 microseconds.

# Programmer's Reference

## InteractWithTeensy(pipe\_TeensyEnd):

Takes: the end of the connection object (pipe) for the Teensy.

This function is passed to a subprocess to run continuously in the background.



Returns: nothing, since it runs indefinitely. However, it sends things via its pipe end to the main GUI process.

**sendToPipe(pipeEnd, intensityBatch, posBatch, prevTime):**

Takes:

- Pipe end, typically the end of the Teensy (ie. sending data to GUI)
- Array of intensity values
- Array of position values
- Time value, designed to be the last time the function was called

Sends intensity and position queues through the given pipe end. Also designed to compute time elapsed since the last time this function was called. Returns two empty arrays and the time the function was called; this is designed to be assigned to the intensity and position queues (to empty them) and for the time to be passed back to the function at the next time the function is called to provide the timing functionality.

Returns:

- Empty array
- Another empty array
- Time this function was called

**clipMarginOnly(array, margin = 0.1):**

Takes:

- Array
- Optional margin value (defaults to 0.1 if none supplied)

Clips the first and last ( $margin / 2$ ) percent of the array, so that in total,  $margin$  percent of the array is removed.

Returns: clipped array

**removeBackground(queue, samplePct = 0.05):**

Takes:

- Array of numbers
- Optional percentage of array to sample

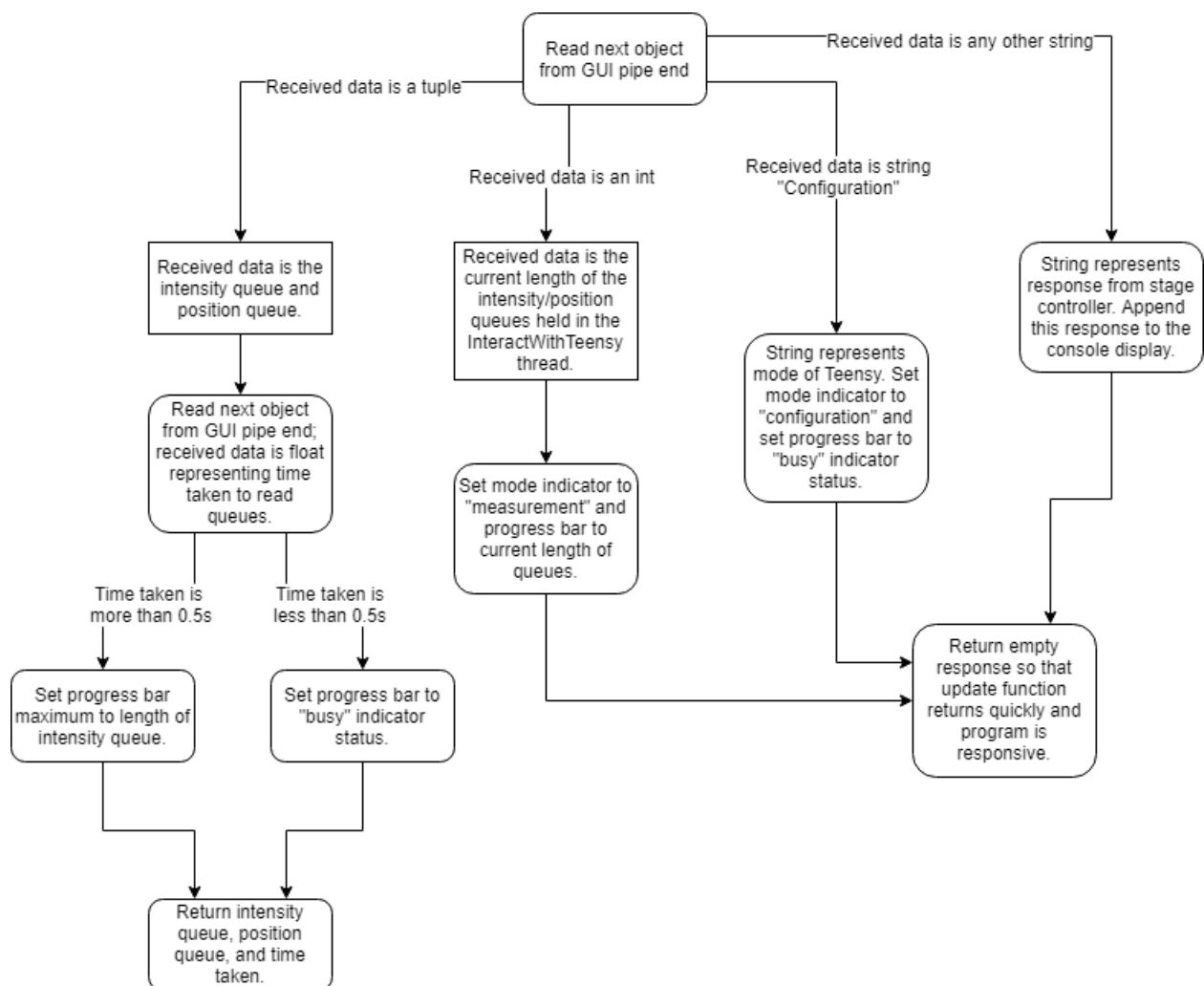
Averages the first  $samplePct$  percent of *queue*, which is assumed to be a baseline background level for the values in the queue. Subtracts this background from every value in the array.

Returns: array with background subtracted from every value in *queue*.

**readPipe(pipe\_GUIEnd, progressBarWidget, teensyModeWidget, consoleWidget):**

Takes:

- the end of the connection object (pipe) for the GUI
- Progress bar widget from GUI
- Teensy mode display widget from GUI
- Console widget from GUI for displaying Teensy/stage controller's responses



**FFT\_to\_spectrum(xsc1,data1):**

Takes:

- Array that represents the amount of time it takes light to travel the extra distances that the mirror moves. Therefore, this array contains information about the position of the stage.
- Array that represents the intensity of the light hitting a sensor. This is expected to vary as the laser beam changes between constructive and destructive interference.

Returns Fourier transform results, converted into an array of wavelengths [nm] and the corresponding intensities.

#### **modifiedGaussian(x, mean, amplitude, deviation, baseline):**

Takes:

- x value to evaluate function at
- Mean of Gaussian function (center; a horizontal translation constant)
- Amplitude of Gaussian function (height of function at mean; a vertical stretch factor)
- Deviation (width of the “bump” of the Gaussian function; a horizontal stretch factor)
- Baseline (value of Gaussian far away from mean; a vertical translation constant)

Returns value of this modified Gaussian function, evaluated at x. Used for fitting to the fringes of a pulsed laser and finding zero-path-difference position.

#### **applyWindow(queueToWindow, parameter = 1):**

Takes:

- An array of numbers, to be passed to Fourier transform.
- Parameter for the windowing function: standard deviation for Gaussian and alpha for Tukey.

Generates a windowing array that has the length of the input array. The windowing array represents the windowing function, which is horizontally scaled so that the first and last value is very small and the center value is close to 1.

Returns a component wise multiplication of the windowing array and *queueToWindow*. (ie. the first value of the windowing array is multiplied with the first value of *queueToWindow* to make the first value of the returned array.)

#### **getCenter(intensityArray):**

Takes: array representing signal from sensor for pulsed laser. We expect fringes to be intense somewhere approximately near the middle already, which is achieved by adjusting the stage's position.

Also uses a symbolic constant, PARTITION\_WIDTH. (Currently set to 50.)

Loops through the entire *intensityArray*, considering PARTITION\_WIDTH values at a time. Finds a partition where the difference between the maximum and minimum values is greatest. The position halfway between this maximum and minimum value is assumed to be the center.

Returns index of the center of the signal array.

#### **clipArraySymmetric(array, center):**

Takes:

- Array to be clipped
- Arbitrary center index of this array, to clip the array around

Returns:

- Array clipped in a way such that the given center index becomes the actual center index of this array.

#### **smoothPosArray(posArray, smoothing = 0, degree = 3, nAverage = 10, stDev = 5):**

Takes:

- Array of numbers, typically representing stage position data
- Optional smoothing parameter to pass to spline constructor
- Optional degrees of spline
- Optional number of points to average in moving average
- Optional standard deviation to use for Gaussian kernel smoothing

Returns:

- xData: an array of integers from 0 to the length of *posArray* - 1 (including these endpoints.)
- Scipy UnivariateSpline object that fits xData and *posArray* as x,y values, after *posArray* is smoothed with kernel smoothing and a moving average.

#### **convertPosToLightTime(posArray):**

Takes: array of recorded positions of the stage, in meters

Performs smoothing on *posArray* with smoothPosArray, as we assume that the posArray represents a stage that is moving continuously.

Returns:

- Array of times equivalent to how long it would take time to travel the smoothed distances
- Array with smoothed positions

- Array with derivative of smoothed positions

### **intensityToPos(reflIntensityArray):**

Takes: array of reference laser intensities

Converts reference intensity signal (expected to be a sine wave, representing alternation between constructive and destructive interference) to an array of distances. This is done via peak detection, and the stage is assumed to have moved a reference laser wavelength as it goes from one peak to another.

Returns:

- Position interpolated for each signal intensity value
- Array of indices where peaks were detected

### **getZeros(array):**

Takes: array of discrete measurements of what is assumed is a continuous signal

Given a set of discrete measurements of what is assumed is a continuous signal, interpolates to find indices where zeros of this signal occur. This is done by making a spline from the points and using SciPy's roots method of the spline.

Returns: how many zeros were found.

### **movingAverage(array, n = 10):**

Takes:

- Array to perform moving average on
- Optional parameter for how many samples to include in each moving average

Moving averages will be centered around each value (ex. The 100th average will be the average of values with index 95-104, assuming  $n = 10$ .) At endpoints, the average will still try to average with a reduced set of data (ex. The value with index 0 in the output array will be the average of the first 5 values in the input array, assuming  $n = 10$ .)

Returns: array containing averaged values.

### **peak\_detect(y, delta, x = None):**

Takes:

- $y$  : intensity data in which to look for peaks

- *delta* : a point is considered a maximum peak if it has the maximal value, and was preceded (to the left) by a value lower by *delta*.
- *x* : correspond x-axis (optional)

Find local maxima in *y*. Converted by Ed Kelleher from MATLAB script at <http://billauer.co.il/peakdet.html>.

Returns:

- Array with two columns.
  - Col1 = indices / the x-values of peaks
  - Col2 = the peak values in *y*

**convertParamsToCounts(start, speed, distance):**

Takes:

- Commanded start position of stage, in mm from one end of motion
- Commanded speed of stage
- Commanded distance for the stage to move in *halfPeriod* seconds, in mm

Returns:

- Count corresponding to center position
- Number of steps to take
- Size each step should be, in counts
- Ticks of stage controller to delay, where each tick is 25.6 microseconds

**getMaxWavelength(wavelengths, intensity):**

Takes:

- Array of wavelengths
- Array of intensities associated with wavelengths

Returns: wavelength with the highest intensity

**getHalfRange(wavelengths, intensity):**

Takes:

- Array of wavelengths
- Array of intensities associated with wavelengths

Splits the array of wavelengths and intensities into two, at the maximum intensity. Therefore, for a typical Gaussian-shaped spectrum, we expect one half to be monotonically increasing with wavelength plotted on the x-axis and intensity on the y-axis, and the other half to be monotonically decreasing. Therefore, we can make a interpolating function that takes intensity



and wavelength as its x,y data (which is opposite to how such graphs are usually plotted--typically wavelength is along the x axis.) If we call these two functions at  $x = 0.5$  (representing half intensity) we get the wavelength corresponding to this half maximum. Therefore, by taking the absolute difference between these two half-maximum wavelengths, we can get a measure of FWHM, assuming that the spectrum is generally in a Gaussian shape.

Returns: absolute difference between wavelengths that correspond to half of maximum intensity.

### **scancommand(pipeEnd):**

Takes: GUI pipe end (values sent in through the GUI end will be received by the Teensy thread at the other end.)

Also takes a symbolic constant, DELAY\_MICROS, representing microseconds to pause at the ends of the motion.

Writes encoded binary ASCII commands to the GUI pipe end, so that the thread interacting with the Teensy can read them and use the Teensy to command the stage controller. The commands are written as follows:

- k0: Turn off the stage
- U 4 ... : Write the starting point of the movement
- U 5 ... : Write the forward step size and number of steps
- U 6 ... : Pause for enough ticks to have a pause DELAY\_MICROS long
- U 7 ... : Write the backward step size and number of steps, which should be the same as commanded in "U 5" but with a negative step size
- U 8 ... : Pause for enough ticks to have a pause DELAY\_MICROS long
- V65535 ... : Run the stage with some number of ticks between movements, indefinitely
- k1: Turn on the stage

Each command is written with some time for the Teensy to pause to allow for communication of each command to the controller to pause. Note that actually writing the commands via serial output to stage controller is done within the InteractWithTeensy function, which runs in its own thread.

Returns nothing. Function call ends after last command is written.

### **setCenterCommand(pipeEnd):**

Takes: GUI pipe end.

Moves the stage to the position indicated in the center input. Commands sent are:

- k0: stage off

- V0: run the stage once
- F ... : move the stage to commanded center position
- k1: stage on

Returns nothing. Function call ends after last command is written.

#### **startcommand(pipeEnd):**

Takes: GUI pipe end.

Sends "k1" to turn on the stage. Returns nothing afterwards.

#### **stopcommand(pipeEnd):**

Takes: GUI pipe end.

Sends "k0" to turn on the stage. Returns nothing afterwards.

#### **saveData(\*args):**

Takes: arbitrary number of arrays, all of the same length.

Writes input arrays as columns into a CSV file. The file is named with the current date and time and placed in the same folder as the program. All of the input arrays are expected to be the same length, so that each row can contain one value from one array. (ie. the third row contains the third value from input array 1, input array 2, input array 3, ...)

Returns nothing. Function call ends after saving file and printing a status update message to the stage console.

#### **sendCommand(pipeEnd):**

Takes: GUI pipe end.

Encodes whatever is in the console command input box into binary ASCII and sends it to the Teensy interaction thread. Instructs Teensy to wait one second after writing this command. Returns nothing afterwards.

#### **sendReadInstruction():**

Takes no arguments.

Essentially acts as a macro for sending the commands “U4”, “U5”, and “V” to the Teensy. By querying these settings from the stage controller, we can read the settings previously commanded to the controller.

Returns nothing. After 4 seconds, triggers readStageSetting.

### **readStageSetting():**

Takes no arguments.

Reads the three most recent lines from the console output, which is expected to be the stage controller’s response to the commands “U4”, “U5”, and “V”, which was sent by sendReadInstruction. Using the information from these responses, fills in the command boxes for stage center, half-period, and scan distance.

Returns nothing, but sets values in the three boxes above.

## **Program flow control functions**

Generally connected to a UI element, and triggered on a click/change. Generally do not take any arguments or return any values, since their purpose is to control the UI.

As a matter of style, these functions are placed immediately before the UI box they are connected to, instead of at the beginning of the program.

### **showIntensityManager():**

Creates global variables (pyqtgraph PlotDataItem objects) used for plotting points. When “show intensity” checkbox is checked, the plots are re-created. When the box is unchecked, the plots are destroyed.

### **updateWithMode():**

Whenever the mode indicator text box changes, enables or disables the stage control UI group. This ensures that the stage settings can only be edited when the Teensy is in configuration mode.

### **CWmodeManager():**

Changes the labels of one box in the status group depending on if CW mode is activated. Also shows/hides right axis label (if CW mode is on, p2 and p2a display the same units, which is intensity.)

### **logPlotManager():**

If the log plot checkbox is checked, enable the log plot and disable the left axis label on the spectrum plot. Otherwise, do the opposite.

Attempts to keep the x-axis the same, but this doesn't seem to work.

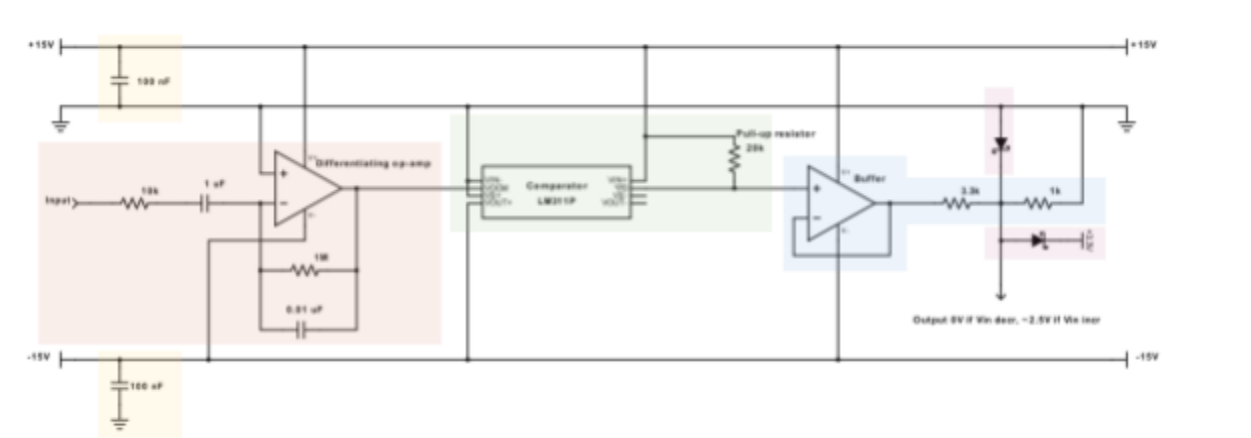
**FFTfilterManager():**

Responds to changes in the FFT windowing function dropdown selector. Sets ranges/enables the parameter input box as appropriate.

# Electronics

## Differentiator

This circuit outputs about 2.5V if the input voltage is increasing with respect to time, and 0V if it is decreasing.



View larger, non-color-coded diagram:

<https://www.digikey.com/schemeit/project/diffcomparator-PR7G430403Q0/>

Component	Function
Op-amp with capacitors and resistors	Outputs the derivative with respect to time of the voltage applied at Input (Vin). Designed so that the position feedback signal from the stage is applied here. With the values of the components shown in the diagram, the differentiator rejects noise of over 15 Hz at the input (otherwise the derivative would be extremely noisy, if we tried to differentiate the noise.)
Capacitors on power rails	Necessary for proper functioning of LM311P comparator. Values were suggested by

	Texas Instruments (manufacturer of chip) in their documentation for LM311P.
Comparator	Compares differentiator signal to 0V, and outputs 0V if signal is negative, and 15V if signal is positive. Recall that the differentiator was applied to take the derivative of the stage's position, so by comparing to a baseline of 0, the comparator tells us if the stage is moving forwards or backwards.
Buffer and voltage divider	Buffers the comparator output signal (otherwise we would be directly connecting +15V through some resistors to ground.) Then, we pass the repeated (buffered) signal through a voltage divider to reduce it to ~2.5V when the comparator outputs 15V. (The zero level remains at zero.)
Schottky diodes	Used to clip the output signal to between 0V and 3.3V before it goes to the Teensy. Note that we expect the signal to be 0-2.5V anyways; this is an extra precaution.