CsPyController Users' Manual

Martin Tom Lichtman

July 6, 2015

Contents

1 Introduction

The CsPyController software was written by MTL to run experiments and collect data for the AQuA (*Atomic Qubit Array*) project. It was designed with generality in mind, and should be usable for a wide variety of physics experiments. It is the successor to CsLabViewController (also by MTL), which was in turn the successor to QCE Suite (by Thomas Henage), and presents many improvements over those codes, including increased flexibility, programmability, ease of use, maintainability, and in particular a built-in optimizer.

This manual demonstrates basic use of the software, including instruments and analyses which are already part of the code. A separate *CsPyControler Programmers' Manual* explains how to extend the code for new instruments and analyses.

2 Architecture

CsPyControler uses a central controller written in Python as the high-level **command center**. This **command center** sequences the experiment, send out commands to instruments, collects data, and runs some analyses. Under normal operation, the users only interacts with the Python **command center**.

Some instruments are controlled by the **command center** directly, however many are controlled by satellite **instrument server** programs. These **servers** may be running on the same computer, or the may run on a different computer. Regardless of location, communication between the **command center** and the **servers** is done using *TCP/IP* internet protocol messaging. This allows the **servers** to be written any language for any operating system. The user must manually start any separate **servers**, but once they are running the **command center** acts as a *TCP/IP client* and all user interaction, setup and control is done from there. The main instruments in use on the AQuA project are on a separate PXI system, controlled by a **server** written in *LabView* running on an embedded *Windows* controller running on that system.

For every experiment, the settings and data are saved into HDF5 files, which is a standardized and well-supported file format which is ideally suited for scientific data collection. Some analyses are done on-the-fly in the **command center**, but any other analysis desired may always be done on the data saved to the HDF5 file, using your scientific computing software of choice (e.g. *Mathematica*, *Matlab*. Or as MTL recommends *Python* with the `ipython`, `h5py`, `numpy`, `scipy` and `matplotlib` packages.)

3 Installation

The CsPyController software is stored in a *GIT* repository on the Saffmanlab `hexagon` server, which makes it easy to archive and maintain version control. The **command center** can be run on any operating system that supports Python, and has been tested on both Windows and Mac OS/X. However, the majority of work has been done on the following configuration, and hence is the only configuration supported:

- Microsoft Windows 8, 64-bit, with Classic Shell

- Python 2.7, 64-bit, via Enthought Canopy
- LabView 2014, 32-bit

3.1 Install GIT

Download the latest Windows binary of *GIT* from `https://git-scm.com/download/win`. Follow instructions there to install GIT. All the default installation choices are acceptable.

3.2 Clone the CsPyController repository

To get a copy of the CsPyController software, you will make a GIT "clone" of the repository. First, map `hexagon` as a network drive by opening the *Windows Explorer*, right clicking *My Computer* or `This PC`, and selecting *Map network drive*. Chose a drive letter (e.g. `Z:`) and then point to the server at `\\hexagon\hexagon D\users`.

Next launch the `Git Gui` from the Start Menu under `All Programs` → `Git` → `Git GUI`. Select *Clone Existing Repository* and for *Source Location* browse to `Z:/Public/software/git_repos/CsPyController.git`. For *Target Directory* install the software wherever you like (e.g. `C:/Users/Hexagon/Documents/git`. A *Standard* copy is sufficient. Click *Clone* to complete the process.

3.3 Install Python

There are many ways to install *Python* and its packages. The recommended and supported method is to use *Enthought Canopy*, which has the easiest package manager available. First, request a free academic license at `https://store.enthought.com/licenses/academic/`. Once your license is confirmed, log in at `https://store.enthought.com/accounts/login/` and download *Canopy 64-bit for Windows* from `https://store.enthought.com/downloads/canopy/win/64/free/`.

Follow instructions there to install *Canopy*.

3.4 Install Python packages

Once *Canopy* is installed, launch the package manager via `Start Menu` → `All Programs` → `Enthought Canopy 64-bit` → `Package Manager`. On servers such as `hexagoncrunch3` you must `Run as administrator` for the package manager to work. Log in using your *Enthough* account. First click `Updates` and then click `Install all Updates`. Then go to `Available Packages` and install the following packages (if they are not already installed):

- enaml (0.9.8-3 or later)
- h5py (2.5.0-3 or later)
- matplotlib (1.4.3-5 or later)
- numpy (1.9.2-1 or later)
- pyaudio (0.2.4-3 or later)
- pypng (0.0.15-1 or later)
- PyQt (4.11.3-1 or later)
- pyserial (2.7-2 or later)
- scikit_learn (0.16.1-2 or later)
- scipy (0.15.1-2 or later)

And for data analysis you will find it very useful to install:

- ipython (3.2.0-1 or later)
- seaborn (0.5.1-9 or later)

Any packages necessary to support these will be automatically install. Not all of these are strictly required to use CsPyController, some of them are convenient to have installed for data analysis.

3.5 Install LabView or LabView Runtime Engine

If you will be using any of the *National Instruments* devices such as *HSDIO, DAQmx or IMAQ* cards, it will be necessary to run the `CsPyController LabView server`. It is not necessary to install a complete copy of LabView (although you can choose to do so), but it will be necessary to install the LabView Runtime Engine (32-bit 2014 SP1 or later), which can be downloaded from `http://www.ni.com/download/labview-run-time-engine-2014/4887/en/`. You may need to create an NI account to download software. The departmental has a site license for most *National Instruments* products, and license number can be obtained from the IT department (e.g. Chad Seys). As of this writing the most recent license number was `H21L20973`. It should be entered when the installers as for `serial number`.

You will also need the runtime or full versions of several LabView packages:

- NI Vision Development Module (2014 SP1 or later)
- NI-845x (14.0 or later)
- NI-DAQmx (14.2.0 or later)
- NI-HSDIO (14.0.0 or later)
- NI-IMAQ (14.5 or later)
- NI-VISA (14.0.1 or later)

Always install LabView before installing packages, so the package installer can auto-detect which versions need support. The packages are available individually from `http://www.ni.com/downloads/drivers/` or you may find most of them in the *NI Device Driver* bundle. Typically it is less time consuming to download the individual drivers, rather than the bundle. Follow the installation instructions for each installer.

3.6 Launch the Python command center

To launch the Python **command center**, open:

`Start Menu→All Programs→Enthought Canopy (64-bit)→Canopy 64-bit command prompt`

Change directories into the `python` subdirectory of your cloned *CsPyController* repository, with the command:

```
cd C:\Users\Hexagon\Documents\git\CsPyController\python
```

where the second argument must be customized to your installation location. Then type:

```
python cs.py
```

Hit enter and the *CsPyController* **command center** will open. The main window has a combo box to select which window your would like to open. Multiple copies of each window may be open and they will stay in sync.

Alternatively, if you set up python to be be in the path of an ordinary Windows command prompt (via the *Canopy* installer or preferences, or manually), then you can launch the **command center** via a batch file. Open *Windows Explorer*, navigate to the directory where you cloned *CsPyController* into, and then into the `python` subdirectory (e.g. `cd C:\Users\Hexagon\Documents\git\CsPyController\python`). Double-click `CsPyController.bat`. You can drag this `.bat` file to the `Start Menu` for quick access.

3.7 Launch the LabView PXI_server

On the machine with the *National Instruments* devices (this may be the same machine), open *Windows Explorer*, navigate to the directory where you cloned *CsPyController* into, and then into the `labview\PXI_server\builds` subdirectory
(e.g. `C:\Users\Hexagon\Documents\git\CsPyController\labview\PXI_server\builds`) and double-click `PXI_server.exe`. This `.exe` can be dragged to your `Start Menu` for easier access.

If you have multiple ethernet cards, and want to restrict incoming connections to a specific network, enter the IP address of the specific ethernet card. Listen for local connections only using `localhost`. Otherwise leave the space blank. This is the address of the machine running **PXI_server**, not the address of the machine running the textbfcommand center.

The IP address of this machine can be found from a command prompt by typing `ipconfig` and noting the line that says `IPv4 Address`. You will need this information later to connect from the **command center**.

The `port` number can be changed to accomodate different network situations, but must match the setting in the `command center` and generally should be left at the default of `9000`.

3.8 Launch the LabView DDS_server

On the machine with the DDS boxes connected via USB (this may be the same machine), open *Windows Explorer*, navigate to the directory where you cloned *CsPyController* into, and then into the `labview\DDS_server\builds` subdirectory
(e.g. `C:\Users\Hexagon\Documents\git\CsPyController\labview\DDS_server\builds`) and double-click `DDS_standalone.exe`. This `.exe` can be dragged to your `Start Menu` for easier access.

If you have multiple ethernet cards, and want to restrict incoming connections to a specific network, enter the IP address of the specific ethernet card. Listen for local connections only using `localhost`. Otherwise leave the space blank. This is the address of the machine running **PXI_server**, not the address of the machine running the textbfcommand center.

The IP address of this machine can be found from a command prompt by typing `ipconfig` and noting the line that says `IPv4 Address`. You will need this information later to connect from the **command center**.

The `port` number can be changed to accomodate different network situations, but must match the setting in the `command center` and generally should be left at the default of `9001`.

3.9 Launch other instrument servers

Depending on what instruments you are using, you may have to launch other instrument servers as well. Use of the Picomotors requires the use of `CsPyController\csharp\PicomotorServer\bin\Debug\PicomotorServer.exe`. The Princeton GigE camera server may usually be run on the same machine as the **command center**, and can be launched from within the **command center** GUI.

3.10 Enter IP addresses into the command center

Take the IP addresses that you noted for the machines running `PXI_server.exe` and `DDS_server.exe` and enter them in the **command center** into the pages for *PXI communication* and *DDS* respectively. If everything is running on the same machine, you may enter `localhost`. These settings save with the experiment settings and will not need to be re-entered.

4 Experiments, Iterations, Measurements and Shots

Let us establish some nomenclature. There are several levels of repetition performed by the software. From big to small, they are *experiments*, *iterations*, *measurements* and *shots*.

An *experiment* is one round of execution with one set of programmed specifications. Generally you run one new *experiment* and get one new results file every time you press *Run* (unless using the optimizer), then you make some change and then run another *experiment*.

An *iteration* is one set of settings. As an *experiment* runs, it can step through different settings as specified by the *independent variables*. Each of these steps is an *iteration*. As detailed later, the *independent variable* settings define the number of *iterations* per *experiment*.

A *measurement* is one execution of the currently programmed experiment sequence. You will generally program an *iteration* to have many *measurements* to gather better statistics. The number of *measurements* per *iteration* is set on the *Experiment* page of the *command center*.

A *shot* is a single camera exposure. Usually you will program two or more *shots* per *measurement* in order to make a before-and-after or time-of-flight comparison. A shot is usually specified in the *Functional Waveform* sequencing, by triggering a digital output pin. For some cameras, the number of expected shots must be entered on their settings page for the purpose of error checking. Every *measurement* in a given *iteration* should have the same number of *shots*.

Note that in almost all cases, lists and arrays in CsPyController are numbered starting from 0. For example, shot 0 is the 1st shot, and shot 1 is the 2nd shot.

5 Variables

One of the strengths of this software package is the ability to work with variables in a flexible and powerful way. There are three places that variables can be specified: *Constants*, *Independent Variables*, and *Dependent Variables*. These variables are stored in a namespace that can be used in almost every other setting in the software.

5.1 Using variables

Settings where variables can be used are called a `Prop` in the code, and are defined by a *name*, *description*, *function*, and *value*. (Some settings such as simple enable/disable, hardware clockrates, etc., are not `Prop`s because it is never necessary for their values to be dynamic.) The *function* is where you enter your desired expression. You can use any valid python syntax. You can reference any defined variable, and even any external package that was imported in the *Constants* and *Dependent Variables*. The entire namespace of `from numpy import *` is made available by default for convenience (so `sin`, `cos`, `pi` and many more are already imported). The value returned by evaluating the *function* must be of the correct type (i.e. interger, float, string) for that `Prop`, otherwise an error will be returned. Keep the **command prompt** window visible to watch for errors, which are usually very descriptive and should be read to see what the problem is. The offending `Prop` will also be highlighted in red.

Evaluation of the variables and `Prop`s occurs at the start of every *iteration*. Upon evaluation, the `Prop` values are updated. Evaluation of the `Prop`s can be forced from the main **command center** window menu under *Evaluation→Update variables throughout experiment*.

The order of evaluation is explained below.

5.2 Constants

Constants are executed first, and only once, at the beginning of the *experiment*. The *constants* are defined in a large multi-line text box on the *Constants and Dependents* page. The *constants* are defined using python code. This could be as simple as, for example:

```
a = 5
readout_time = 7
```

Or it can be more complex, involving function definitions, imports, and much more. The namespace of `from numpy import *` is available, as well as anything else the user chooses to import. The entire namespace that results from the execution of this code is preserved and available in the rest of the variable evaluation process.

See Appendix A for a model example of the AQuA **constants**.

5.3 Independent Variables

Independent Variables are evaluated second, once at the beginning of each *iteration*. The independent variables have a more structured entry format. You can add a new *independent variables* by hitting the + button at the top of the page. The new variable will be inserted at the position in the list specified by the integer spin box at the top left. To remove a variable, set the integer spin box to the index of the variable you would like to remove, and hit the - button.

Each *independent variable* is defined by a function, which can use anything defined in the *constants* namespace. This function must evaluate to something that can be cast to a 1D *numpy array*. In other words, it must either be a scalar or a 1D list. Useful functions include: `linspace(a, b, n)` to make `n` evenly spaced points between `a` and `b` inclusive, and `arange(a, b, step)` to make points with spacing `step` from `a` to `b`. You can also enter just a single value (i.e. `0.271`) or a list of specific values (i.e. `[3, 0.2, 4.9, 5.0]`.

The evaluated functions are immediately shown. Check this to make sure you have entered the *independent variable* as intended.

Every *iteration*, only one scalar value is used for each *independent variable*. The values are iterated through in a series of nested loops, with the inner-most loop being at the top of the list, and the outer-most loop at the bottom of the list. The number of iterations is defined by the product of the length of each *independent variable* evaluation. *Independent variables* of length 1 do not contribute to the number of *iterations*. You do not need to have any *independent variables*, but there is always at least 1 *iteration*.

After this evaluation, the *independent variable* names are in the variable namespace, and they have a single scalar value associated with them. For the purposes of later calculation, they are not lists or arrays, no matter how large a list or array was used to define them.

The remaining fields are used for the *optimizer*. In particular, note that the *min* and *max* fields have **no** effect unless you are using the optimizer. It is allowed to assign a value outside of this range during normal a normal iterative experiment run.

5.4 Dependent Variables

Dependent variables are executed 3rd. The *dependent variables* are defined by a large multi-line text box, just like the *constants*. The difference is that *dependent variables* are evaluated at the start of each *iteration* but after the *independent variables*, so they can implement functions that rely on both *constants* and *independent variables*. The *dependent variables* then update as the *iterations* are stepped. Everything defined by the *constants*, *independent variables* and *dependent variables* is then stored in a namespace that is available to all later `Prop` function evaluations.

See Appendix B for a model example of the AQuA **functional waveforms**.

5.5 Exclude variables from HDF5

Everything in the variable namespace is stored to the results *HDF5* file for each iteration. However there are types of objects that cannot be stored (and usually you would not want to store these), such as imported modules and function definitions. To avoid seeing errors related to failure to store these variables, put them in the exclusion list at the top of the *Constants and Dependent Vars* page, titled *Variables Not to Save to HDF5*. Put the names of the excluded variables, separated by commas, with no spaces.

6 Program Controls

6.1 Running an experiment

To run an experiment, simply go to the main window menu and click *Experiment→Reset and Run*, or hit `ctrl+r` from the main window.

You may also select *Experiment→Reset*, which creates the experiment files but does not begin the run. Then you can begin the run with *Experiment→Run/Continue*. It is rare that you would want to use this.

6.2 Pausing

There are several different ways to plan for an experiment to be paused. If enabled (on the *Experiments* page) there is an audible alert when paused. *Run/Continue* or `ctrl+g` is used to continue an experiment that has been paused. Note that the *Pause After Measurement* and *Pause After Iteration* settings will remain checked and will keep pausing the experiment after each measurement or iteration, until they are manually unchecked. Usually if you have used *Pause After Measurement* to do a temporary pause, you will uncheck the setting before selecting *Continue*.

Experiment→Pause→After Measurement waits until the end of the current measurement to pause. If the experiment is between measurements, but already evaluating for the next measurement, it will not pause until after the 1st measurement of the coming iteration. *Pause After Measurement* is generally the best way to do an immediate pause, as it does not compromise the experiment status.

Experiment→Pause→After Iteration waits until the end of the current iteration to pause. It is useful if there is a piece of hardware that you need to manually adjust between iterations, or if you would like to end the experiment early but have all the iterations taken be complete.

Experiment→Pause→Now sets the status to Paused. It is not the best way to pause a running experiment and may compromise the experiment status. It is useful if the experiment has been prematurely stopped, and you would like to set the status back to *Paused* so that you may then *Continue*.

6.3 Uploading

The experiment files are uploaded to the server (if enabled on the *Experiments* page) at the end of the *experiment*. However, if you would like to finish the experiment prematurely, first pause it (*Pause After Measurement* would be the most graceful in this case) and then select *Experiment→End and Upload*. This will do any final calculations and then upload the experiment.

If there is some error that is preventing the experiment from reaching the upload stage, but you still want to preserve the data, just use the *Experiment→Upload*. If all else fails you may find the experiment directory and copy it to the server yourself.

6.4 Closing the Program

The program will not quit until allow windows have been closed, by clicking the X on the titlebar of each window. You can use *File→Quit* or `ctrl+q` to close the main window. You can also forcibly end the whole program by hitting `ctrl+c` in the **command prompt** window. Make sure you have saved any changes because they are **not** auto-saved on exit.

6.5 Experiment Page

The *Experiment* window can be brought up using the page selector on the main **command center** window. This page has important master settings for the whole experiment:

- `Status`: The running/paused/idle status of the experiment.
- `Pause after iteration`: When selected, the experiment will pause after each iteration.
- `Pause after measurement`: When selected, the experiment will pause after each measurement.
- `Reload settings after pause?`: If selected, all the hardware settings will be resent when the experiment is continued after a pause. It is usually a good idea to use this when continuing after a `Pause on error`, but it slows things down unnecessarily when continuing from a `Pause after measurement`.
- `enable sounds`: Enable or disable playing sounds on pause, error and experiment success.
- `start each instrument in a separate thread`: Click to enable multi-threading of the `start` command to each experiment. Could increase experiment speed, but not usable if the experiment start order is critical due to the start trigger setup.
- `Save Data?`: Check to allow saving of data, and give the path to the directory where daily directories will be created.
- `Save Settings?`: Automatically save the experiment settings when the experiment is run.
- `Save separate notes.txt?`: Click to save the notes in a separate `notes.txt` file, in addition to within `results.hdf5`.
- `Save 2013 style files?`: Click to save the ASCII data files compatible with Larry's Mathematica analysis code. Disable also overrides the `Save as PNG` and `Save as ASCII` *Hamamatsu* camera settings.
- `Copy Data to Network?`: Check to allow copying of the entire experiment directory to the server after the experiment is complete, and give the path to the server. Requires the server to be setup as a mapped drive.
- `Experiment description suffix for filename`: The name of the experiment, to be appended to the timestamp filename (e.g. `suffix` gives `2015_07_05_16_53_12_suffix`). No spaces or special characters.
- `Measurement Timeout`: The amount of time the measurement is allowed to proceed in the **PXI_server** and **DDS_server** before raising an error.
- `Measurements per Iteration`: The number of measurement repeats per one set of iteration settings.
- `E-mail on error/completion?`: Check to allow the software to send an e-mail when there is an error or when the experiment completes, and give the e-mail address to send to.
- `Notes`: Text about the specific experiment that is useful when looking at old results.
- `Time status and progress`: Read only. Various statistics are given about the experiment progress. When using the optimizer, the progress will read high than 100% after the first experiment. The completion time prediction takes into account the deadtime between measurements and iterations, and becomes more accurate as the experiment progresses.

7 HDF5 files

Settings and data are stored in *HDF5* files. *HDF5*, or *Hierarchical Data Format 5* is a standard file format, controlled by the HDF Group. It is the answer to all your prayers. HDF5 files store numerical data efficiently in a binary format, but they contain sufficient metadata so that there is never any confusion about what the data type is. They can handle multi-dimensional arrays. The various data saved into the *HDF5* file can be placed in a tree format to keep it well organized.

You can easily see what data are in an *HDF5* file by using the HDFView Java program. This program is also useful to edit settings files by deleting parts that you do not want, or to combine settings from two or more files. **CsPyController** does not have the capability to load partial settings, as **CsLabViewController** did, because you can do this using *HDFView* in a far more powerful and flexible way.

HDF5 files can be accessed from all the numerical analysis software (*Mathematica*, *Matlab*, *Python*), but the best way to work with them is using the `h5py` package in Python because it allows *HDF5* data to be loaded as `numpy` arrays, and also it is extremely fast.

HDF5 files have internal paths to label the location of each dataset. The experiment settings, including every variable setting, instrument setting, and analysis setting, are stored under `/settings/experiment`. The settings are saved automatically into the results file at the start of each experiment (if the `Save Settings?` box is checked on the *Experiment* page of the **command center**), so that there is always a good record of what experiment was run. The settings can be saved manually from the **command center** menu at *File→Save*. Any time the settings are saved, either automatically or manually, the default settings, stored in `settings.hdf5` in the **CsPyController/python** program directory.

Every time an experiment is run, a new directory is created, named with the timestamp of the start of the experiment and the suffix defined on the *Experiment* page (`yyyy_mm_dd_hh_mm_ss_suffix`). This directory is placed inside a daily directory (created if it does not already exist), inside the data directory specified on the *Experiment* page. The data file is named `results.hdf5`. Within the HDF5 file, data for each iteration is stored at `/iterations/0`, `iterations/1`, etc. Every iteration node then has a place for analyses (i.e. `/iterations/0/analysis`) such as loading data. The data for each measurement is then stored at `/iterations/0/measurements/0`, `/iterations/0/measurements/1`, etc. Within these nodes you will find all the raw and processed data for the experiment.

8 Optimizer

The **optimzer** allows CsPyController to automate improvements to your experiment settings. The **optimizer** functions by running your **experiment** many times, and making changes to some or all of the **independent variables**, in a way that minimizes a **cost function**.

8.1 Saved Data

Each loop of the **optimizer** is considered another **experiment**. In the HDF5 file, the optimizer loops are collected under `/experiments/0`, `/experiments/1`, etc. These nodes contain virtual links to the **iterations** within these experiments, which are re-numbered starting from `0` for each loop, even as the labels continue to increase monotonically under the `/iterations` section of the HDF5 file.

The analyzed cost function data is saved in the HDF5 file under `/analysis`.

8.2 Setup

8.2.1 Iterated Variables

To set up the optimizer, first set up an experiment either as you usually would, or in such a way as to amplify sensitivity to the optimal parameters (for example: adding an optical depumping phase after optical pumping, or adding a trap drop phase when trying to lower atom temperature). This experiment

may be just one iteration, or it may be multiple iterations. If iterating over some variable, make sure that the `optimize?` box is **un**checked, otherwise the **optimizer** will control it and it will not step as desired.

8.2.2 Optimized Variables

Next, create **independent variables** for each setting that you would like to **optimizer** to be able to adjust. For each of these, check the `optimize?` box. Assign some non-zero value to `initial step (abs)` which defines the initial step size for the **optimizer**. Choose a step size large enough to illuminate the dependence of the cost function on the variable, but not so large to make convergence take overly long. `(abs)` indicates that this is an absolute step size, not a percent or fraction. Next, enter the `end tolerance (abs)`. The **optimizer** will end when none of the variables change by more than their `end tolerance`. Finally, enter a `min` and `max`. These values are safety limits. They do not influence the normal behavior of the **optimizer**, but if the **optimizer** puts a variable value outside of the `min`/`max` range the actual output is capped.

8.2.3 Optimizer Settings

Then, on the *Optimizer* page, turn on the `enable override`. To activate the **optimizer**, you need to enable both on this page, and each **independent variable** individually. Choose an `update method`:

- `Nelder-Mead`: This method, also known as the *Simplex* method, is an excellent general purpose optimization algorithm, because it does not require prior knowledge of the scale of the local features. This is the most recommended method to use.
- `Weighted Nelder-Mead`: Similar to the Nelder-Mead algorith, but instead of doing simple reflections of the simplex in the average point, takes the cost function value at each point into account to make reflections in the centroid instead.
- `Gradient Descent`: Makes a number of measurements around the initial point to determine the gradient, then moves in the direction of the steepest downward slope. This method converges fast if you know the scale of the local features, but is not useful if you do not know the scale. The gradient is measured over steps determined by the **independent variable** step sizes. The size of the move down the slope is defined by the `initial gradient step size` parameter set on the *Optimizer* page. The algorithm termination is defined by the `end when step size is less then` parameter on the *Optimizer*, not by the individual `end tolerance`s on the *Independent Variables* page.
- `Genetic`: This algorithm makes random moves, keeps them if they are an improvement, and discards them if they are not. It's convergence is generally too slow to be useful.

8.2.4 Cost Function

Finally, you must define the **cost function**, on the *Optimizer* page. The cost function box is treated as python code, which is then executed. The variable `experimentResults` is available in this namespace, and its value is the HDF5 node for the most recent completed experiment. Use `experimentResults` to gain access to the results of the last **experiment** run, which is what the cost function should be based on. Finally, after calculating the cost function, assign it to `self.yi` which is where the code will access it.

Here is an example that maximizes the retention:

```
# get the first iteration
iterationResults = experimentResults['iterations/0']
# get the retention for all regions of interest
retention_all = iterationResults['analysis/loading_retention/retention'].value
# average over all sites (ignore sites with NaN due to no loading)
```

```
retention_avg = numpy.nanmean(retention_all)
# assign the cost function, use a minus sign because the optimizer is a minimizer
self.yi = -retention_avg
```

See Appendix D for a model example of the AQuA **cost functions**.

8.3 Feedback

The graph on the *Opimizer* page shows the cost function in the top subplot, and each variable being optimized in the subplots below. Whenever the optimizer finds a new lowest cost, it sets the function for every **independent variable** being optimized to match those settings. After running the optimizer, the experiment is therefore already set to the best settings. You may have to copy those settings to **constants** if you plan to remove the **independent variables**. Also, note that while the `results.hdf5` file for the optimizer run has all the information needed to read out the best settings, the settings stored under `/settings/experiment` are (as always) saved at the beginning of the run, not the end. Therefore, if you were to load settings from that HDF5 file, it would set up the controller to re-run the **optimizer** from its starting point, not its best point. This is by design as the settings in an HDF5 file should always allow you to recreate an experiment exactly. To save the new best settings after an optimizer run, manually save or just run the next experiment.

During the optimizer run, you will often see non-optimal iterations as the optimizer finds its way. The last point is not necessarily the best point.

Typically, the more variables you put into the optimizer, the more of a speedup advantage you gain from doing a multi-dimensional optimization, as all the variables are adjusted simultaneously. However, there is a linearly increasing cost to starting up the optimizer with many variables, as it has to evaluate the initial simplex or gradient.

9 Functional Waveforms

The **functional waveforms** are how the sequencing for the `HSDIO digital output`, `Analog Output`, and `DAQmx digital output` are specified. Formerly, a table was used. This system is superior because:

- it allows building sequences both in series (timewise) and in parallel (channelwise)
- it allows events to be referenced to either an absolute or relative times
- many different time references can be created so that the event times refer to the most relevant other event in the sequence
- sequences can be built out of many different nested levels, so that low level functions can be defined once, and high level functions can be mixed and matched to define an experiment
- it eliminates the need to specifically calculate the length of any sequence
- allows implementation of complex behaviors like automatic Gray coding
- waveforms are more compact (a "sparse" implementation), and evaluation is much faster
- repetition eliminated, so changes can be made across all waveforms sharing some function
- nest waveforms to reuse code
- backend code is actually simpler in many ways
- view a universal graph of HSDIO and AO waveforms, across the entire experiment

- waveforms are readable, commentable, and make a lot more sense

The way the **functional waveform** system works is to start by providing only 3 basic functions:

- `HSDIO(t, channel, state)`: sets one HSDIO `channel` to `state` (high = True or 1, low = False or 0) at time `t`
- `AO(t, channel, value)`: sets one Analog Output `channel` to voltage `value` at time `t`
- `DO(t, channel, state)`: sets one DAQmx digital output `channel` to `state` (high = True or 1, low = False or 0) at time `t`
- `label(t, text)`: labels the **functional waveforms graph** at time `t` with the label `text`, which allows the graph to be understood in terms of the phases of an experiment

These functions can then be used in the *Functional Waveforms* window, which has a multi-line text box that allows execution of arbitrary code. This code has access to the namespace of variables defined by the **constants**, **independent variables**, and **dependent variables**. It is re-evaluated every **iteration**.

9.1 Setup

Begin by importing the `add_transition` functions:

```
HSDIO = experiment.LabView.HSDIO.add_transition
AO = experiment.LabView.AnalogOutput.add_transition
DO = experiment.LabView.DAQmxDO.add_transition
label = experiment.functional_waveforms_graph.label
```

Now proceed to define useful functions. Best practice is that every function, always, should take in the start time as the first parameter, and return the end time when it is done.

```
def pulse(t, channel, duration):
    """A square pulse on one channel for a limited duration."""
    HSDIO(t, channel, True)  # turn the channel on
    t += duration  # wait until the duration passes
    HSDIO(t, channel, False)  # turn the channel off
    return t  # return the end time

def ramp(t, channel, v1, v2, duration):
    """Sweep one analog output channel from v1 to v2."""
    t_list = linspace(t, t+duration,100)  # make 100 time steps
    v_list = linspace(v1, v2, 100)  # make 100 voltage steps
    for t, v in zip(t_list, v_list):
        # step through each value and assign it to the AO channel
        AO(t, channel, v)
    return t+duration  # return the end time
```

Defining a function does not execute it. So you can keep many different functions in your text for various uses. However, at the bottom of the **functional waveforms** text box, after your functions have been defined, you must actually call the functions so that they execute:

```
# define some channels
laser = 3
camera = 5
t = 0  # reset the time
t += pulse(t, laser, 5)  # do a 5 ms laser pulse
t += pulse(t, camera, 30) # take a 30 ms exposure
```

By using the `t +=` syntax, the time variable `t` keeps incrementing, so you do not have to do any other math to get the sequence timing correct. Events can be specified to be simultaneous, or even out-of-order, to achieve the desired sequence. The `HSDIO`, `AO` and `DAQmxDO` instruments will then calculate their waveform output automatically.

See Appendix C for a model example of the AQuA **functional waveforms**.

9.2 Functional Waveforms Graph

The *Functional waveforms Graph* page shows a highly customizable graph of the `HSDIO`, `AO` and `DAQmxDO` behavior.

- `enable`: Enables or disables the graphing.
- `HSDIO`: A list of the HSDIO channels to plot. Valid formats include `range(32)` for all channels from 0 to 31, `[13,14,15]` for just several channels, or `[11]` for just one channel (note the brackets are required to make a list, even for one channel), or blank for none.
- `HSDIO ticks`: Enable or disable vertical markers and x-axis tick labels for the `HSDIO` transitions
- `AO`: A list of the AO channels to plot. Valid formats include `range(5)` for all channels from 0 to 4, `[2,3,4]` for just several channels, or `[1]` for just one channel (note the brackets are required to make a list, even for one channel), or blank for none.
- `AO ticks`: Enable or disable vertical markers and x-axis tick labels for the `AO` transitions
- `DO`: A list of the DAQmx Digital Output channels to plot. Valid formats include `range(5)` for all channels from 0 to 4, `[2,3,4]` for just several channels, or `[1]` for just one channel (note the brackets are required to make a list, even for one channel), or blank for none.
- `DO ticks`: Enable or disable vertical markers and x-axis tick labels for the `DO` transitions
- `labels`: Toggle whether to show numerical x-axis tick labels (unchecked) or the text labels that were specified using the `functional_waveforms_graph.label` method.
- `plot min`: The minimum time to plot.
- `plot max`: The maximum time to plot.
- `units`: The time units for the plot. (1 = seconds, .001 = milliseconds)
- `AO scale`: By default, the AO plots have the space between two channel plots equal to one volt. This setting sets the number of volts/division. (A bigger number shrinks the plots vertically, a smaller number expands them.)

10 Instruments and Analyses

Each instrument has a separate settings page, accessible via the combo box selector on the main window of the **command center**.

10.1 Analog Input

The Analog Input instrument controls an DAQmx AI task via the LabView **PXI_server** and requires the **PXI_server** program to be running. Recorded data is saved to `results.hdf5` at `/iterations/#/measurements/#/da`

- `enable`: Check to enable the instrument.
- `sample_rate`: How often to take data, in Hz. (Variables okay, must evaluate to float.)
- `source`: The source for the AI channels, which can be determined through the NI Measurement and Automation Explorer. (Variables okay, must evaluate to string.)
- `samples_per_measurement`: How many samples to take after the start trigger. (Variables okay, must evaluate to int).
- `waitForStartTrigger`: If `False`, start via software trigger at the beginning of the measurement. If `True`, wait for the trigger specified in `triggerSource`. (Variables okay, must evaluate to bool.)
- `triggerSource`: The source for the start trigger, which can be determined through the NI Measurement and Automation Explorer. (Variables okay, must evaluate to string.)
- `triggerEdge`: Choose to trigger on the rising or falling edge. (Variables okay, must evaluate to string "Rising" or "Falling".)
- `channels`: Used to keep track of channel descriptions. Use the + and - buttons to add or remove an item at the listed index.
- `Analog Input Filter`: Choose to reject certain measurements based on AI levels.
  - `enable`: Enable or disable this filter.
  - `What to filter`: Program the filter with a list of the format `[(channel, samples list, low, high), (channel, samples list, low, high), ...]`. The samples list is averaged over. For example `[(1,[33,34],.12,.15),(2,range(10),-.5,-.4)]` will only pass measurements where channel 1 reads between .12 V and .15 V when averaged over samples 33 and 34, and channel 2 reads between -.5 V and -.4 V when averaged over samples 0 through 9.
  - `Filter level`: Choose the action to take if the filter fails.
- `Analog Input Plots`: Show the AI levels after every measurement. This graph shows one data point per channel per measurement, averaged over the selected samples. It does not show an oscilloscope trace vs samples within a measurement.
  - `enable`: Enable or disable the plots.
  - `What to plot`: Program the plot with a list of the format `[(channel, samples list), (channel, samples list, ...]`. The samples list is averaged over. For example `[(1,[33,34]),(2,range(10))]` will only plot channel 1 averaged over samples 33 and 34, and channel 2 averaged over samples 0 through 9.
  - `clear`: The AI plot will persist from experiment to experiment, until cleared using this button.

10.2 Analog Output

The Analog Input instrument controls an DAQmx AO task via the LabView **PXI_server** and requires the **PXI_server** program to be running. Sequencing of the AO events is done using the **functional waveforms**, where the method `experiment.LabView.AnalogOutput.add_transition(time, channel, voltage)` is available to set a given `channel` to some `voltage` at some `time`. The `add_transition` calls

need not be done in order, and can be looped to create ramps or arbitrary waveforms. Unless otherwise specified, channels revert to 0 volts at t = 0, but hold their values after the last specified transition until the beginning of the next measurement.

- `enable`: Enable or disable this instrument.
- `physicalChannels`: The source for the AO task, which can be determined using the NI Measurement and Automation Explorer. (Variables okay, must evaluate to string.)
- `number of channels`: Specify the total number of channels, which must match the number listed in `physicalChannels`.
- `channel descriptions`: A list of strings describing each channel, used in the **functional waveforms graph**. The format is `['name1','name2','name3']` with the number of names matching the `number of channels`.
- `minimum`: The lowest voltage that the hardware will be allowed to produce (generally cannot be lower than -10 V). (Variables okay, must evaluate to float.)
- `maximum`: The highest voltage that the hardware will be allowed to produce (generally cannot be higher than +10 V). (Variables okay, must evaluate to float.)
- `clockRate`: Clockrate in Hz. (Variables okay, must evaluate to float.)
- `units`: The time units to be used in the **functional waveforms**. (1 = seconds, .001 = milliseconds). (Variables okay, must evaluate to float.)
- `waitForStartTrigger`: If `True`, wait for a hardware start trigger. If `False`, start via software trigger at the beginning of the measurement. (Variables okay, must evaluate to bool.)
- `triggerSource`: The source of the start trigger, if used. Can be found using the NI Measurement and Automation Explorer. (Variables okay, must evaluate to string.)
- `triggerEdge`: Choose to trigger on the rising or falling edge. (Variables okay, must evaluate to string "Rising" or "Falling".)
- `exportStartTrigger`: Choose if you want a signal to be sent to the PXI bus when this instrument starts. In the AQuA experiment, all other PXI instruments are triggered off this signal. (Variables okay, must evaluate to bool.)
- `exportStartTriggerDestination`: The LabView path to export the start trigger signal. Can be determined using the NI Measurement and Automation Explorer. (Variables okay, must evaluate to string.)
- `useExternalClock`: If `False`, use the internal clock, if `True` use an external clock from the port specified in `externalClockSource`. (Variables okay, must evaluate to bool.)
- `externalClockSource`: The LabView path for the external clock. Can be determined using the NI Measurement and Automation Explorer. (Variables okay, must evaluate to string.)
- `maxExternalClockRate`: An upper bound for the external clock rate. Does not have to be exact. (Variables okay, must evaluate to float.)

10.3 Andor

Controls an Andor camera, directly from CsPyController, so no external servers are necessary. Requires prior installation of the Andor USB drivers.

- `enable`: Enable or disable this instrument.
- `video mode`: Shows an auto-triggering camera image on this page. This mode cannot be used during an experiment.
- `stop video`: Stops the video mode.
- `EMCCDGain`: Set the electron multiplier gain value. (Variables okay, must evaluate to int.)
- `preAmpGain`: Set the pre-amp gain value. (Variables okay, must evaluate to int.)
- `exposureTime`: The camera exposure, when in edge trigger mode, time in seconds. (Variables okay, must evaluate to float.)
- `trigger mode`: Select the trigger mode. `Edge` mode starts the exposure on the trigger edge, but then the exposure time is determined by the `exposureTime` setting. In `Level` mode, the expose starts on the trigger edge, but then the exposure continues as long as the trigger stays high.
- `shotsPerMeasurement`: The expected number of camera triggers, for error checking.
- `shot`: Which shot to show on this page.

10.4 Arroyo

For control and readout from Arroyo temperature controllers. Not fully implemented.

10.5 Constants and Dependent Variables

Space to assign variables. See Section 5 for info.

10.6 Counters

Measurement of digital pulse counts via a DAQmx Counter task via the LabView **PXI_server**. Requires the **PXI_server** program to be running. Recorded data is saved to `results.hdf5` at `/iterations/#/measurements/#`

- `enable`: Enable or disable this instrument.
- `index`: Use the + and - buttons to add or remove a channel specification at the index shown.
- `counter source`: The LabView path for this counter channel. Can be determined using the NI Measurement and Automation Explorer. (string)
- `clock source`: The LabView path for the gate signal for this counter channel. Can be determined using the NI Measurement and Automation Explorer. (string)
- `clock rate`: If the `clock source` is internal, use this to specify the clock rate in Hz. Unused for external clocks. (float)

10.7 DAQmx Digital Output

The DAQmx Digital Output instrument controls lower-speed digital output through a DAQmx DO task via the LabView **PXI_server**. It requires the **PXI_server** program to be running. Sequencing of the DO events is done using the **functional waveforms**, where the method `experiment.LabView.DAQmxDOt.add_transition channel, state)` is available to set a given `channel` to `state` (True = high, False = low) at some `time`. The `add_transition` calls need not be done in order. Unless otherwise specified, channels revert to low at t = 0, but hold their values after the last specified transition until the beginning of the next measurement.

- `enable`: Enable or disable this instrument.
- `resourceName`: The LabView path of the DAQmx DO output card. Can be determined using the NI Measurement and Automation Explorer. (Variables okay, must evaluate to string.)
- `clockRate`: The clock rate in Hz. (Variables okay, must evaluate to float.)
- `units`: The time units used in the **functional waveforms**. (1 = seconds, .001 = milliseconds) (Variables okay, must evaluate to float.)
- `Start Trigger)`: Define when this instrument starts its output.
  - `waitForStartTrigger`: If `False` the output starts according to a software trigger at the beginning of the measurement. If `True`, wait for a hardware trigger. (Variables okay, must evaluate to bool.)
  - `source`: The LabView path to the hardware start trigger. Can be determined using the NI Measurement and Automation explorer. (Variables okay, must evaluate to string.)
  - `edge`: Choose the rising or falling edge of the hardware start trigger. (Variables okay, must evaluate to string `"rising"` or `"falling"`.)

10.8 DC Noise Eaters

This instrument communicates with Saffmanlab custom DC (or Slow) Noise Eater boxes, which were designed by Alex Gill. This instrument can handle any number of boxes. Communication is over USB, and requires the *Prop Plug* adapter to interface to the *Parallax Propeller* microcontroller. See the DC Noise Eater manual on the Saffmanlab Wiki for more info.

- `enable (instrument)`: Enable or disable all the DC Noise Eater boxes.
- `add/remove box`: Use the + and - buttons to add or remove a box, at the location defined by the spinbox index. The spinbox also controls which box's settings are visible.
- `enable (box)`: Enable or disable this specific box.
- `channel index`: Change this spinbox index to view the 3 channels available on each box.
- `Set`: Allow the settings entered on the computer to overwrite those on the noise eater. You will want this uncheck initially, when you first start using computer control of an existing box.
- `Get`: Allow the settings on the noise eater to overwrite those on the computer.
- `mode`: Sets the operation mode for the channel.
  - `off`: The channel output is off.
  - `run`: Normal PI feedback operation.
  - `idle-hi`: 10 V output

  - `idle-med`: 5 V output
  - `idle-lo`: 0 V output
  - `ramp`: Scan output from 0 to 10 V.
- `trigger warning?`: Set the warning if this channel's absolute error signal is greater than `limit range`.
- `invert`: Invert the channel PI behavior.
- `integration time`: How long the box should integrate the current signal from the photodiode from this channel, in microseconds.
- `trigger number`: The DC noise eater has one input for multiple photodiodes. This selects which pulse on Trigger 2 corresponds to this channel.
- `measurements to average` How many past measurements should be rolling averaged for the error signal, to smooth out noise. (Note: set this to 1, it is not a good feedback principle to use this. Use the I integration setting instead.)
- `Kp`: The proportional feedback constant.
- `Ki`: The integration feedback constant.
- `setpoint`: The input level (in units of mV boxcar integrator output for the given `integration time` that the DC Noise eater tries to hold in `run` mode.
- `average`: Read-only. The averaged input level in mV.
- `error`: Read-only. The difference between the `average` and the `setpoint` in mV.
- `V in`: Read-only. The input voltage in mV.
- `V out`: Read-only. The output voltage in mV.
- `warning`: Read-only. Whether or not the `error` exceeded the `limit range`.
- `DC Noise Eater Filter`: Provides the ability to discard measurements based on the DC Noise Eater data.
  - `enable`: Enable or disable the filtering.
  - `What to filter`: A list of tuples that defines the filter, in the form `[(box,channel,variable,low,high),(`
    where `box` and `channel` define which signal is being monitored, `low` and `high` define the upper and lower bounds that will pass this filter, and variable is a code which defines which part of the data we are filtering on (some of these are more useful than others, usually you will use code 11 for the error signal):
    * 0: mode
    * 1: trigger warning?
    * 2: limit range
    * 3: invert
    * 4: integration time
    * 5: trigger number
    * 6: measurements to average
    * 7: Kp

    - 8: Ki
    - 9: setpoint
    - 10: average
    - 11: error
    - 12: vin
    - 13: vout
    - 14: warning

    For example, `[(0,1,11,-60,60)]` specifies that box 0, channel 1 will be filtered on the error signal being within ±60 mV.
  - `filter level`: The action to take if the filter fails.

- `DC Noise Eater Plots`: Plot the data returned by the DC Noise Eater. This graph updates every measurement, and does not reset between experiments, unless you manually clear it with the `clear` button.
  - `enable`: Enable or disable the plotting.
  - `What to plot`: Program what to plot using a list of tuples in the format `[(box,channel variable),(box,channel,variable),...]` where `box` and `channel` specify the signal choice, and `variable` uses the same code as `variable` for the `DC Noise Eater Filter` (given above. For example `[(0,0,11),(0,1,11),(0,2,11)]` plots the error signal for all three channels on box 0.

10.9 DDS

This instrument controls any number of Saffmanlab DDS boxes, each consisting four Analog Devices AD9910 Eval Boards. It requires **DDS_server.exe** to be running on the machine to which the DDS boxes are connected by USB. See the DDS box manual on the Saffmanlab Wiki for more info.

- `open connection`: Opens a connection to the **DDS_server**. The connection opens automatically when the experiment is run, and it is not usually necessary to use this button. However it can be useful for diagnostics or error recovery.
- `close connection`: Manually close the connection to the **DDS_server**. Not usually necessary, but useful sometimes for error recovery.
- `IP address`: The IP address of the machine where the **DDS_server** is running. You can use `localhost` if it is on the same machine.
- `TCP port`: The TCP/IP port used for communication with the **DDS_server**. Must match the setting in the **DDS_server**. Usually the default of `9001` is fine.
- `timeout`: The amount of time to allow for communication with **DDS_server** before it gives up and raises an error. This setting should be in seconds, however it seems that there is an error in the `python.socket` documentation, and this setting is really in milliseconds. Therefore a value of at least `100` is recommended.
- `enable (instrument)`: Enable or disable communication with the **DDS_server**.
- `initialized`: Read only. Indicates if the **DDS_server** has been initialized yet.

- `Get DDS Device List`: Requests the USB ID tags of all the DDS boxes running on the **DDS_server** machine. Pressing this button populates the `select USB device reference` combo box, but does not actually use those values without further action. This function is a very useful feature to determine which USB boxes are communicating properly in case of error, because the malfuncitoning boxes will not show up on the list. If running, pause the experiment before using this.
- `Initialize and Load`: This initializes the DDS boxes and loads the current settings onto them. Do not re-initialize when you just want to change settings, because this can unlock some lasers. Initialization is done automatically at the beginning of an experiment if necessary. If running, pause the experiment before using this.
- `Load`: Load the current settings onto the DDS boxes. Use this when the boxes have been previously initialized, and you just want to manually load the current settings. It is less disruptive than `Initialize and Load`. If running, pause the experiment before using this.
- `DDS boxes`: Use the combo box to select which DDS box you would like to view. Use the + and - buttons to add a box, or remove the active box.
- `enable(box)`: Enable or disable communication with a specific box.
- `Description`: A description for the box, which is used to populate the box combo selector.
- `select USB device reference`: This is a convenience feature for getting the right USB device reference. Use the `Get DDS Device List` button to update this list. After selecting the appropriate USB device reference for this box, click the `select USB device reference` button to copy this string to the `NI USB-8451 device reference`. Merely selecting a choice without clicking the button does nothing.
- `NI USB-8451 device reference`: The USB ID tag for this box.
- `DIO port`: Leave this set to 0.
- `serialClockRate`: Leave this set to 1000.
- `channels`: Every box has four channels (i.e. four eval boards). All channels are shown below at the same time.
- `channel`: The settings for a particular channel, numbered 0 to 3.
  - `channel parameters`: The settings that apply to the entire channel.
    - `description`: Description for the channel.
    - `power`: Turn the RF output for this channel on (True) or off (False). (Variables okay, must evaluate to bool.)
    - `refClockRate`: (Not editable.) The clock rate of the reference. Cannot be changed from 1 GHz.
    - `fullScaleOutputPower`: Scaling for the max output power, in dBm. (Variables okay, must evaluate to float.)
    - `RAMenable`: Enable (True) or disable (False) the RAM mode. (Variables okay, must evaluate to bool.
    - `RAMDestType`: A code that defines the RAM mode: 0: Frequency, 1: Phase, 2: Amplitude, 3: Polar. (Variables okay, must evaluate to int.)
    - `RAMDefaultFrequency`: The default output frequency for RAM mode in MHz. (Variables okay, must evaluate to float.)

    - `RAMDefaultAmplitude`: The default output amplitude for RAM mode in dBm. (Variables okay, must evaluate to float.)
    - `RAMDefaultPhase`: The default output phase for RAM mode in radians. (Variables okay, must evaluate to float.)
  - `profiles`: The eight settings profiles for this channel, numbered 0 to 7. Use the combo box to select a profile to edit.
    - `description`: A description for this profile, used to populate the profile selector combo box.
    - `frequency`: The (non-RAM mode) output frequency for this profile, in MHz. (Variables okay, must evaluate to float.)
    - `amplitude`: The (non-RAM mode) output amplitude for this profile, in dBm. (Variables okay, must evaluate to float.)
    - `phase`: The (non-RAM mode) output phase for this profile, in radians. (Variables okay, must evaluate to float.)
    - `RAMMode`: The RAM mode to use, if active. Choose from 0: Direct Switch, 1: Ramp Up, 2: Bidirectional Ramp, 3: Continuous Bidirectional Ramp, 4: Continuous Recirculate, 5: Direct Switch 2, 6: Direct Switch 3. (Variables okay, must evaluate to int.)
    - `ZeroCrossing`: Define if the Ramp ram modes cross zero. (Variables okay, must evaluate to bool.)
    - `NoDwellHigh`: For the non-recirculating RAM modes, define if the state at the end stays at the last setting (False) or returns to the beginning (True). (Variables okay, must evaluate to bool.)
    - `FunctionOrStatic`: True to evaluate the RAM mode steps using `RAMFunction` or False to specify the steps using the `RAMStaticArray`.
    - `RAMFunction`: A function of one variable `x`, which is evaluated to define the RAM mode steps. Must be specified as a string, can use any python syntax. The variable `x` is then swept from `RAMInitialValue` to `RAMStepValue * RAMNumSteps`. Each step is held for `RAMTimeStep`. (Variables okay, must evaluate to string.)
    - `RAMInitialValue`: The starting value for the RAMFunction. (Variables okay, must evaluate to float.)
    - `RAMStepValue`: How much to increase `x` on each step. (Variables okay, must evaluate to float.)
    - `RAMTimeStep`: How long to hold each step, in microseconds. (Variables okay, must evaluate to float.)
    - `RAMNumSteps`: How many steps to make on the RAM function, before ending or recirculating. Max is 1024 total for all profiles.
    - `RAMStaticArray`: Defines the RAM stepping when `FuncitonOrStatic` is False. Use the + and - buttons and the index to add or remove steps. Total number of steps is 1024 for all profiles.
      - `f/phi/A`: The frequency, phase or amplitude (depending on the mode) for this step.
      - `Mag`: The magnitude for this step, when using polar mode.

- `connected`: The state of the connection to the **DDS_server**.
- `DDS error`: The error log reported back from **DDS_server**.

10.10 Filters

A number of different filters that can be used to dump data that does not fit certain criteria. There are also filters on some of the individual instrument pages. Additional filters can be defined, as documented in the *CsPyController Programmer's Manual*.

10.10.1 TTL Filters

Can be used to dump data when a TTL signal is low, such as for the laser lock monitors. Requires the **PXI_server** to be running.

- `enable`: Enable or disable this filter.
- `lines`: The LabView path for the TTL input. Can be determined using the NI Measurement and Automation Explorer. Lines are checked before and after each measurement, and if any line is low for either check, the filter fails.
- `filter level`: The action to take if the filter fails.
- `Status`: Read only. The pass/fail status of the filter with info about which TTL line failed.

10.10.2 Region of Interest Sum Filter

Filters on the brightness of the Hamamatsu regions of interest.

- `enable`: Enable or disable the filter.
- `boolean filter expression`: A boolean expression which evaluates to True or False and determines the action of the filter. The variable `t` is provided which is an array containing the region of interest sums, and has shape `shots x region`. All python logical expressions are valid here. For example `t[0,27]>64000 and t[0,41]>89000` only passes measurements where region 27 has a summed brightness greater than 64000 and region 41 has a summed brightness greater than 89000.
- `filter level`: The action to take if the filter fails.
- `Status`: Read only. The pass/fail status of the filter.

10.10.3 Drop First N Measurements Filter

Drop the first `N` measurements of every iteration, to allow transients to settle.

- `enable`: Enable or disable this filter.
- `N`: The number of measurements to drop.
- `filter level`: The action to take if the filter fails.

10.11 Gaussian ROI

Automatically identifies the regions of interest for the *AQuA* atom array. Fits the mean Hamamatsu image to a 7x7 grid of gaussians. The degrees of freedom are grid position in x and y, grid tilt, grid spacing, gaussian width in two directions, gaussian tilt, and background level. The gaussians are normalized, and used as an image mask that acts as a region of interest. Every new image that comes in is multiplied by each of these 49 gaussians, giving 49 arrays which each isolate one region of interest. Each of these arrays is then summed, which gives 49 region of interest values.

This analysis can perform the initial identification of the regions of interest, and that behavior can then be turned off so that only the region of interest sums continue to be evaluated on each measurement.

- `enable`: Enable or disable the analysis.
- `perform gaussian grid fit`: Enable or disable the finding of new regions of interest. This does not control whether or not the new ROIs are adopted.
- `automatically use ROIs from valid fits`: Newly calculated regions of interest can be adopted manually with the `use these ROIs` button, or they can be adopted automatically by selecting this option. If the new ROIs need to be used in other analyses done on-the-fly this same iteration, then they will need to be adopted automatically.
- `calculate gaussian ROI sums`: Use the gaussian ROIs to calculate region of interest sums. This is generally always desired and can be done even when a new ROI fit is not being calculated.
- `shot` Which camera shot to use to do the ROI fit.
- `subtract background?`: Subtract the background image (chosen on the *Live Images* page) before performing the ROI fit.
- `subtract background from sums?`: Subtract the background image before calculating the ROI sums.
- `multiply sums by photoelectron scaling?`: Multiply the sums by a calibration of *counts/photoelectron*, which is specified on the *Hamamatsu* page.
- `clean up image with ICA?`: Use independent component analysis to filter the image before finding new regions of interest. ICA identifies sets of correlated pixels. By using the 49 most correlated sets, we get a good identification of the atom sites with good rejection of background noise.
- `use these ROIs`: If the new grid fit regions of interest were not adopted automatically, click this to manually adopt them.
- `initial guess`: Give the grid fit algorithm an initial guess for the `top`, `left`, `bottom` and `right` pixel positions of the grid.
- `graph`: The graph shows the raw mean image, ICA filtered mean image, the initial guess for the grid, the best fit for the grid, and the 1 sigma contours of the gaussian ROIs superimposed on top of the raw or ICA filtered image.

10.12 HSDIO Digital Output

The HSDIO instrument controls an HSDIO DO task via the LabView **PXI_server** and requires the **PXI_server** program to be running. Sequencing of the HSDIO events is done using the **functional waveforms**, where the method `experiment.LabView.HSDIO.add_transition(time, channel, state)` is available to set a given `channel` to some `state` (True = high, False = low), at some `time`. The `add_transition` calls need not be done in order. Unless otherwise specified, channels revert to low at t = 0, but hold their values after the last specified transition until the beginning of the next measurement.

- `enable`: Enable or disable this instrument.
- `resourceName`: The LabView path of the HSDIO card, which can be determined using the NI Measurement and Automation Explorer. If using more than one HSDIO card, separate them with a comma (e.g. `'PXI1Slot4,PXI1Slot5'`). (Variables okay, must evaluate to string.)
- `clockRate`: Clockrate in Hz. (Variables okay, must evaluate to float.)

- `units`: The time units to be used in the **functional waveforms**. (1 = seconds, .001 = milliseconds). (Variables okay, must evaluate to float.)
- `hardwareAlignmentQuantum`: The minimum sample resolution, which is a specification for particular HSDIO hardware. The HSDIO cards on the *AQuA* PXI chassis have a hardware alignment quantum of 1, which means any length sequence can be specified. The HSDIO card on the Square Cell PCI computer has a hardware alignment quantum of 2, which means waveforms and wait times must be an even number of samples. (Variables okay, must evaluate to int.)
- `number of channels`: Specify the total number of channels, in multiples of 32, which should match the number of cards.
- `waitForStartTrigger`: If `True`, wait for a hardware start trigger. If `False`, start via software trigger at the beginning of the measurement. (Variables okay, must evaluate to bool.)
- `triggerSource`: The source of the start trigger, if used. Can be found using the NI Measurement and Automation Explorer. (Variables okay, must evaluate to string.)
- `triggerEdge`: Choose to trigger on the rising or falling edge. (Variables okay, must evaluate to string "rising" or "falling".)
- `Script Triggers`: An array of up to four programmable triggers which can be used in the HSDIO waveforms. This is not implemented for **functional waveforms**. Use the + and - buttons to add or remove a trigger at the index given by the spin box.
  - `description`: A helpful description of the trigger.
  - `id`: The name of the trigger as it will be used in the HSDIO waveforms. (Variables okay, must evaluate to string "ScriptTrigger0", "ScriptTrigger1", "ScriptTrigger2" or "ScriptTrigger3".)
  - `source`: The hardware source for the trigger. Can be determined using the NI Measurement and Automation Explorer. (Variables okay, must evaluate to string.)
  - `type`: If this is an edge or level trigger. (Variables okay, must evaluate to string "edge" or "level".)
  - `edge`: If using an edge trigger, if this should be the rising or falling edge. (Variables okay, must evaluate to string "rising" or "falling".)
  - `level`: If using a level trigger, if this should be a high level or low level. (Variables okay, must evaluate to string "high" or "low".)
- `channels`: A list of the channels.
  - `description`: A description that is used on the **functional waveforms graph**.
  - `active`: Enable or disable this channel. (Variables okay, must evaluate to bool.)

10.13 Hamamatsu

The settings for the Hamamatsu EMCCD camera.

- `enable`: Enable or disable the camera.
- `saveAsPNG`: Save each shot taken as a lossless `.png` file in the experiment directory. Also requires `Save 2013 style files?` to be checked on the *Experiment* page. It is not recommended to use this, because all the image data is available in a more stable and more efficient format in `results.hdf5`. (Variables okay, must evaluate to bool.)

- `saveAsASCII`: Save each shot taken as a `.txt` file in the experiment directory. Also requires `Save 2013 style files?` to be checked on the *Experiment* page. It is not recommended to use this, because it is an extremely inefficient way to store image data.

- `forceImagesToU16`: All Hamamatsu C9100-13 data is unsigned 16-bit integers, however on certain CameraLink acquisition cards it reports as signed 16-bit. Set this to True to cast the data to U16. (Variables okay, must evaluate to bool.)

- `EMGain`: Set the electron multiplier gain. (Variables okay, must evaluate to int 0 to 255.)

- `analogGain`: Set the analog gain. (Variables okay, must evaluate to int 0 to 4.)

- `exposureTime`: When using edge or internal trigger mode, sets the exposure time in seconds. (Variables okay, must evaluate to float.)

- `scanSpeed`: Choose the readout speed. (Variables okay, must evaluate to string "Slow", "Middle" or "High".)

- `lowLightSensitivity`: Set the low light sensitivity mode. (Variables okay, must evaluate to string "Off", "5x", "13x" or "21x".)

- `externalTriggerMode`: When using an external trigger, select if you want an edge trigger, where the exposure time is determined by the `exposureTime` setting, or a level trigger where the exposure time is determined by the length of the trigger pulse. (Variables okay, must evaluate to string "Edge" or "Level".)

- `triggerPolarity`: Select if you are looking for a high or low trigger. (Variables okay, must evaluate to string "Positive" or "Negative".)

- `externalTriggerSoruce`: Select the input source for the external trigger. (Variables okay, must evaluate to string "Multi-Timing I/O Pin", "BNC on Power Supply", or "CameraLink Interface".)

- `cooling`: Turn the TEC cooling on or off. The appropriate DIP switches must also be set on the camera, see the Hamamatsu manual for details. (Variables okay, must evaluate to string "On" or "Off".)

- `fan`: Turn the camera fan on or off. The appropriate DIP switches must also be set on the camera, see the Hamamatsu manual for details. (Variables okay, must evaluate to "On" or "Off".)

- `scanMode`: Choose the pixel mode for the readout. Normal to read out all pixels. Super pixel to bin together groups of pixels. Sub-array to read out a smaller region than the entire image sensor. (Variables okay, must evaluate to "Normal, "Super pixel" or "Sub-array".)

- `photoelectronScaling`: A calibration value for the number of counts per photoelectron. Used to scale ROI sum data. (Variables okay, must evaluate to float.)

- `subArray`: Left, Top, Width and Height settings for use in the `Sub-array scanMode`. Speeds up image readout by reducing the portion of the sensor used. (Variables okay, must evaluate to int in steps of 16 up to sensor size.)

- `superPixelBinning`: If selected in `scanMode`, choose how many pixels to bin together. (Variables okay, must evaluate to string "1x1", "2x2" or "4x4".)

- `frameGrabberAcquisitionRegion`: The image size can be cropped in the CameraLink frame grabber card. Set the Left, Top, Right and Bottom crop borders. (Variables okay, must evaluate to int in steps of 1 up to the sensor size.)

- `shotsPerMeasurement`: The number of camera triggers to expect per measurement, for error checking.

10.14 Histogram

Plots a histogram of the region of interest sum data for the current iteration, live updated every measurement.

- `enable`: Enable or disable the plot.
- `what to plot`: Use a list of tuples to specify what to plot in the format `[(shot,region),(shot,region)]`. For example `[(0,17),(1,17)]` will plot region of interest 17 in both shots 0 and 1.

10.15 Histogram Grid

Plot a histogram of region interest data every iteration. Shows histograms for each of a 7x7 grid of regions of interest at the same time. This analysis can find a brightness cutoff for atom loading data by fitting a gaussian above and below test cuttoffs, and finding the best fit to two gaussians. In the plot, the 0 atom signal is shown in blue and the 1 atom signal is shown in red, with gaussian fits over-layed on top. If `Save Data?` is selected on the *Experiments* page, then the plots for every shot are saved to disk as `.pdf` files in the experiment directory.

- `enable`: Enable or disable the plots.
- `shot to display`: Which shot to show on screen. All shots are still analyzed for saving to `.pdf` files.
- `roi type`: Choose whether to use the `gaussian` or `square` regions of interest, which are defined on their own settings pages.
- `calculate new cutoffs`: Choose whether to calculate the best fit cutoffs. This alone is not enough for the new cutoffs to be used.
- `automatically use new cutoffs`: If checked, newly calculated cutoffs will be adopted.
- `cutoff shot mapping list`: This list specifies which shots to glean the cutoffs from. For example, if there are two shots, then `[0,0]` always uses shot 0 for the cutoffs, `[0,1]` uses each shot for its own cutoff data, and `[1,1]` always uses shot 1 for the cutoffs.
- `use new cutoffs`: If not automatically adopting newly calculated cutoffs, you can use this button to manually adopt the most recently calculated cutoffs.

10.16 Image Browser

You can use this window to browse through all the shots taken so far for the entire experiment.

- `Show ROIs?`: Click to show the region of interest outlines over the images.
- `independent variable values`: Every independent variable is shown with a combo box for the possible values for that variable. Select the values that you want for each independent variable, and the images available will be culled to only represent those settings.
- `measurement`: Select the measurement you wish to view, for the iteration selected through the `independent variable values`.
- `shot`: Select the shot to show, for the selected measurement and iteration.

10.17 Iterations Graph

A plot that shows the averaged region of interest sum data versus iterations. The scale on the x axis is iterations, and the scale on the y axis is region of interest sum brightness for the Hamamatsu camera data.

- `shots and regions to graph`: Specify the shots and regions to graph using a list of tuples in the format `[(shot,region),(shot,region)]`. For example `[(1,22),(1,23)]` plots regions 22 and 23 from shot 1.
- `update every measurement?`: (deprecated) Allows the plot to update live every measurement throughout the iteration. Deprecated because it was too slow.
- `enable`: Enable the plot.
- `draw connecting lines`: Draw lines between the points.
- `only add data that has passed loading filter?`: If checked, only plot data that has passed the `Region of Interest Sum` filter on the *Filters* page.
- `draw error bars`: Draw the 1 sigma error bars for the standard deviation of the mean.
- `ymin` and `ymax`: Use to set the vertical range of the plot. The plot range will be set automatically if these are blank.

10.18 Laird Temperature Controllers

For readout from Laird temperature controllers. Requires **box_temperature_server.py** to be running separately.

- `enable`: Enable or disable communication with this instrument.
- `IP address`: The IP address of the machine where **box_temperature_server.py** is running.
- `port`: The TCP/IP port number of the **box_temperature_server.py**.

10.19 Live Images

This analysis shows the live incoming images for the Hamamatsu camera. The left panel shows the most recent raw image, while the right panel shows the live updating average image for the current iteration.

10.19.1 Most Recent Image

- `Show ROI?`: Overlay the square ROI outlines on the plot.
- `shot`: Select which shot to show from the most recent measurement.
- `subtract background`: Subtract the background from the image. Does not affect stored data.

10.19.2 Mean Image

- `enable`: Enable or disable the mean image calculation and plotting.
- `Show ROIs?`: Overlay the square ROIs onto the image.
- `shot`: Select which shot to show the mean image for.

- `ymin` and `ymax`: Use to set the intensity scale for the mean image. The scale will be set automatically to the min/max for the visible shot if these are left blank. The *Most Recent Image* will scale according to the mean image.
- `subtract background`: Subtract the background from the mean image. This does not affect the stored data.
- `use this image as background`: Click to use the currently showing mean image as the background. The selected shot will be used, but it is applied as the background image to all shots.

10.20 Measurements Graph

A plot that shows the region of interest sum data with a datapoint for every measurement. The scale on the x axis is # of measurements.

- `enable`: Enable the plot.
- `shots and regions to graph`: Specify the shots and regions to graph using a list of tuples in the format `[(shot,region),(shot,region)]`. For example `[(1,22),(1,23)]` plots regions 22 and 23 from shot 1.
- `clear`: The data points do not clear every iteration of experiment. The continue to accumulate until cleared using this button.

10.21 PI Piezo

Controls the Physik Instrumente (PI) 3 axis P-611.3 Nanocube piezo stages. Requires the **PXI_server** to be running on the machine where the PI piezo controller racks are connected via USB. Can communicate with any number of PI controllers. The spin box index selects which controller settings are visible. Use the + and - buttons to add or remove a controller at the index selected.

- `enable (instrument)`: Enable or disable this instrument.
- `description`: A useful description for each controller.
- `enable (controller)`: Enable or disable just a specific controller.
- `serial number set`: Enter the serial number of the controller defined by the visible settings.
- `id and serial number read`: Read only. A readout of the serial number from the controller, to confirm communications are working.
- `setServo`: Turn the closed loop servo feedback on for each channel. (Variables okay, must evaluate to bool.)
- `setPosition`: The (closed or open loop) position setting for each channel, in μm. (Variables okay, must evaluate to float.)
- `read from piezo`: Read only. Status readout for each channel.

10.22 PXI Communication

The *HSDIO*, *Analog Output*, *DAQmx Digital Output*, *Analog Input* and *Counter* instruments all require the **PXI_server** to be running. Communication for these instruments is controlled on the *PXI Communication* page. The settings here are:

- `open connection`: Opens a new connection to **PXI_server** if there is not already one. This is not usually necessary because this happens automatically if needed in the experiment sequence.
- `update settings`: Pushes the current settings to the **PXI_server**. This can be useful to enact some manual change in the equipment state, or the *cycle continuously* setting. If updating a setting you will want to force evaluation first (from the main window menu).
- `close connection`: Forcibly close the TCP/IP connection to the **PXI_server**. This is useful if there has been some unrecoverable communications error and you would like to renew the connection.
- `enable`: A master enable for the TCP/IP communication for all these instruments.
- `IP address of LabView system`: The IP address of the machine where the **PXI_server** is running. You may use `localhost` if it is on the same machine.
- `communications port`: This must match the port selection in **PXI_server**. The default of `9000` usually works fine.
- `cycle experiment continuously even when not taking data`: If unchecked, the **PXI_server** will only start to take a measurement when the **command center** requests it. If checked, this setting causes the **PXI_server** to keep taking measurements, even when they have not been requested. This allows faster data taking, because the next measurement can be started before the previous one has been analyzed, and when the **command center** finally does request a measurement, it is given the results of the most recent completed measurement that it has not yet received. Checking this setting also allows the experiment duty cycle to stay constant, keeping the temperature constant.
- `timeout`: This puts a master timeout on the activity of the PXI instruments. Make sure this value is longer than the requested measurement sequence.

10.23 PID

A PID controller to replace the DC Noise Eater boxes, by using the Analog Input and Analog Output instruments. Not yet fully implemented.

10.24 Picomotors

Control any number of Newport Picomotors. Requires **picomotor_server.exe**, a code written in C#, to be running on the machine where the Picomotors are connected via USB. This will be replaced in the future by direct HTTP communication to the Picomotor controllers.

- `enable`: Enable or disable this instrument.
- `IP address`: The IP address of the machine where the **picomotor_server** is running.
- `port`: The TCP/IP port in use by **picomotor_server**. Usually 11000.
- `+ and - buttons`: Use the index to change which motor settings are visible, and use the + and - buttons to add or remove a motor at the specified index.

- `serial number`: The serial number of this controller where this motor is attached, which lets **picomotor_server** direct the communications.
- `motor number`: Specify the connection number of this motor on the controller.
- `position`: The desired position to be set. These are open loop devices, so an exact position is not guaranteed. (Variables okay, must evaluate to int.)

10.25 Princeton Camera

Control for the Princeton EMCCD GigE camera. Added by Donald Booth, modeled from the Andor and Picomotor code. Requires a server program to be running, which can be launched using this GUI. Horizontal and vertical sliders change the image cropping.

- `video mode`: Start showing internally triggered video images.
- `stop video`: Stop triggering video images.
- `Start Server`: Launch the external server program on this machine.
- `enable`: Enable or disable this instrument.
- `AdcEMGain`: The electron multiplier gain for the camera. (Variables okay, must evaluate to integer.)
- `exposureTime`: The length of the exposure when using edge trigger mode. (Variables okay, must evaluate to float.)
- `trigger mode`: Select Edge to allow the exposure time to be defined by the `exposureTime` setting. Select Level to let the length of the trigger determine the exposure time.
- `shotsPerMeasurement`: The expected number of camera triggers per measurement. Used for error checking. (Variables okay, must evaluate to int.)
- `shot`: Select which shot to show in the plot window.

10.26 RF Generators

This instrument allows control of Agilent/HP RF generators over GPIB. Three models of generators are defined, but the communication protocol is very similar for all HP RF generators, and other models may work with the defined communications. For each model, there is an array of settings that allows any number of boxes of that type to be used. This instrument requires the **PXI_server** to be running on the machine where the GPIB connection is made.

- `enable`: Enable or disable this instrument.
- `HP83623A` Use the spinbox to view settings for different generators. Add any number of boxes by using the + and - buttons to add or remove a setting at the index shown on the spinbox.
  - `enable`: Enable or disable communication to this specific generator.
  - `description`: The purpose of this specific generator.
  - `GPIB channel`: The GPIB id number of this generator, set using the controls on the generator. GPIB id numbers must be unique.
  - `frequency`: The frequency setting for this generator, in MHz. (Variables okay, must evaluate to float.)
  - `power`: The power setting for this generator, in dBm. (Variables okay, must evaluate to float.)

  - `RFoutput`: Turn the RF output for this generator on or off. (Variables okay, must evaluate to bool.)
  - `externalTrigger`: Select whether or not to use an external trigger to control output of the generator. (Variables okay, must evaluate to bool.)

- `HP8662A` Use the spinbox to view settings for different generators. Add any number of boxes by using the + and - buttons to add or remove a setting at the index shown on the spinbox.
  - `enable`: Enable or disable communication to this specific generator.
  - `description`: The purpose of this specific generator.
  - `GPIB channel`: The GPIB id number of this generator, set using the controls on the generator. GPIB id numbers must be unique.
  - `frequency`: The frequency setting for this generator, in MHz. (Variables okay, must evaluate to float.)
  - `power`: The power setting for this generator, in dBm. (Variables okay, must evaluate to float.)

- `HP83712B` Use the spinbox to view settings for different generators. Add any number of boxes by using the + and - buttons to add or remove a setting at the index shown on the spinbox.
  - `enable`: Enable or disable communication to this specific generator.
  - `description`: The purpose of this specific generator.
  - `GPIB channel`: The GPIB id number of this generator, set using the controls on the generator. GPIB id numbers must be unique.
  - `frequency`: The frequency setting for this generator, in MHz. (Variables okay, must evaluate to float.)
  - `power`: The power setting for this generator, in dBm. (Variables okay, must evaluate to float.)

10.27 Ramsey

This analysis is a plot of retention vs iterations, which also has a built in fit function to fit to a the function $amplitude * \cos(2pift)e^{t/decay} + offset$. The purpose is that this can be used in the optimizer to maximize some oscillation frequency. This could be used not just Ramsey experiments, but also Rabi or any oscillation. Results are saved into the HDF5 file under `/experiments/#/analysis/Ramsey/frequency` and can be accessed in the optimizer cost function via `experimentResults['analysis/Ramsey/frequency']`.

- `enable`: Enable or disable this analysis.
- `draw error bars`: Draw the 1 sigma error bars for the standard deviation of the mean.
- `roi`: Choose which region of interest to analyze.
- `time variable name`: Give the name of the variable that represents time. This will be used as `t` in the fit function.
- `fit`: The fit parameters: `amplitude`, `frequency`, `offset` and `decay`. Give a `guess` for each parameter. The results will be updated into the `fit` box.

10.28 Report

This analysis is used to print out any diagnostic information about the variables. This can be used to make sure the variable iteration behavior is performing as expected, or to do some useful calculation. There are two mutli-line text boxes: The `Constant Report` is evaluated after the **constants**, once per experiment, and has access only to the **constants** namespace. The `Variable Report` is evaluated after the **dependent variables**, and so has access to the full namespace of **constants**, **independent** and **dependent variables**. To define the report, use python syntax that evaluates to a string. The entire result of each box must evaluate to one string, so the entire box must be one python statement. Concatenation of multiple lines is possible using parenthesis to contain the entire statement and + to denote string concatentation.

See Appendix E for a model example of the AQuA **report** code.

10.29 Retention Analysis

This analysis gives the loading, retention and reloading statistics for each region of interest. This data is used in other analyses and so it is usually necessary to have this enabled.

- `enable`: Enable or disable this analysis.
- roi type: Choose whether to use the *square ROI* or *gaussian ROI* cutoffs.

10.30 Retention Graph

This graph uses the retention data calculated in *Retention Analysis* to plot retention vs iteration.

- `regions to graph`: A list of tuples that specifies what to plot, using the format `[region1, region2]`. For example `[2,7]` plots regions 2 and 7, which `range(49)` plots all 49 regions.
- `enable`: Enable or disable the plot.
- `draw connection lines`: Draw lines between the data points.
- `draw error bars`: Draw the 1 sigma error bars of the standard deviation of the mean.
- `only add data that has passed loading filter`: Only use data that passed the *Region of Interest Sum Filter* on the *Filters* page.
- `ymin` and `ymax`: Use this to set vertical plot limits. If left blank, the vertical range is determined automatically.

10.31 Square ROI

A table for manually specifying square regions of interest for the Hamamatsu camera data. The number of regions can by changed by modifying the ROI list length in the HDF5 file. The plot on this page shows a digital representation of atom loading for the listed regions and cutoffs. The positions of the regions themselves may be viewed on the *Live Images* page.

- `enable`: Enable or disable this analysis.
- `left, top, right, bottom`: Specify the position of each region.
- `threshold`: Specify a cutoff threshold where a summed region of interest brightness under this level is considered no atom, while above this level is considered 1 atom.

11 Known bugs

11.1 Independent Variables

Do not use multiple copies of the *Independent Variables* page, because items added and removed will not be synced to other pages. Also, the index shown on each variable listing is not necessarily correct after variables have been added and removed. It is the position in the list that matters, not the printed variable index. Closing and reopening the window will correct these problems.

The order of the independent variables is important, as this defines which variable is the inner loop, and therefore the data order of the iterations. However, the independent variables list is alphabetized in the HDF5 file, and therefore loading an old experiment, or quitting and restarting the program, will alphabetize the list, which can change the order.

11.2 HDF5

The iterations and measurements in the HDF5 files, although labeled `0`, `1`, etc, are ordered alphabetically, not numerically. This means that when analyzing data you must sort the data properly according to the labels, and not take the default order. Otherwise you you will get
`0,1,10,11,12,13,14,15,16,17,18,19,2,20,...`

11.3 Regions of Interest

Several of the analyses which are no longer used were programmed to use the older square regions of interest, and have not been updated to enable the use of the gaussian regions of interest. This includes the *Region of Interest Sum Filter* and *Iterations Graph*.

Appendices

These appendices which show the variable code from the *AQuA* project, are provided as examples and best-practice references.

A AQuA Constants

```
# settings updated 2015-07-02
# These constant settings do not need to be commented out
# because they will be overwritten by the independent and dependent variables.

### experiment flags ###

background = False # Use when taking a background image. Turns off 2D and 3D MOT during idle,
    loading and PGC1
alignment = False # changes dds rydA profile 3 to lower power when true
doGates = True # Set to false to change delay_before_excitaiton and switching_delay to 5 ms,
    when you DON"T want blockade.
doRamsey1038 = False # changes rydA waveform leaving the traps on, turns ryd 459 off in DDS
doRamsey459 = False  # turns 1038 off in dds
do1038Trap = False # sets Pre_Readout_Time to 0, turns on 1038 (780 still on)
doPatternLoad = False # turns on first blow away to do patterned loading if true
all780off = False # sets the 780 bits to always off


### packages ###
```

```
import numpy as np


### SI Prefixes ###

tera=1e12; giga=1e9; mega=1e6; kilo=1e3; milli=1e-3; micro=1e-6; nano=1e-9; pico=1e-12;


### physics constants ###

c0=299792458 # m/s
eps0=8.854187817e-12 # F/m
alpha=7.2973525698e-3
kB_SI=1.3806488e-23 # J/K
hyperfine_splitting_6s1_2 = 9.19263e9
hyperfine_splitting_7p1_2 = 377.4e6
gamma_6p3_2 = 2*pi*5.1e6
gamma_7p1_2 = 2*pi*1e6
alpha_780_cgs = -247e-24
alpha_780_SI = alpha_780_cgs*4*pi*eps0*1e-6
saturation_intensity = 27 # W/m^2


### MOT ###

MOT_loading_time = 480.250866709
MOT_frequency = 107.682685945 # profile 0
MOT_power = -3.12639959015 # profile 0

hyperfine_frequency = 268.425729262 # profile 0
hyperfine_power = -0.253770318123 # profile 0

MOT_2D_time = 429.924189099 #controlled by a shutter. needs time to open and close. bounces upon
     opening
trap_on_time_during_loading = 435.982310345
MOT_drop_time = 65.2040961701


### PGC 1 ###

PGC_1_time = 6.11868915721
PGC_1_frequency = 91.4499601693 # profile 1
PGC_1_power = -6.82121127689 # profile 1
PGC_1_hyperfine_frequency = 278.5 #not in any waveform
PGC_1_hyperfine_power = 0 #not in any waveform
MOT_to_PGC_B_field_delay = .6
trap_780_delay_for_PGC_1 = 4.28533651393


### PGC 2 ###

PGC_2_time = 8.53133176334
PGC_2_frequency = 81.9991546641 # profile 4
PGC_2_power = 0.213636966912 # profile 4
PGC_2_hyperfine_frequency = 278.5 #not currently in any waveform
PGC_2_hyperfine_power = 0 #not currently in any waveform
Readout_to_PGC_2_B_field_delay = .001
```

```
### Trapping ###
SHG_780_freq = 75.8393240905 # controls DDS AOM RF frequency
SHG_780_power = -3.01422252233 # controls DDS AOM RF power. Best output is at -2.5 dBm, so do
    not go higher.


### Blowaway ###

# talks to HP 8665A on GPIB18
blowaway_freq = 219.986031065 # 219 MHz in traps, 227 MHz on resonance
blowaway_power = -11.8722125912 #dBm
blow_away_time = 0.21308523234
blow_away_time2=.05 if doPatternLoad else 0

### Readout ###

Readout_time = 49.3480613968
Readout_time2=28.61*1.5
Readout_frequency = 93.7598022702 #profile 3
Readout_power = -6.03238305528 #profile 3
Readout_hyperfine_frequency = 278.0 # not in any waveform
Readout_hyperfine_power = 0 # not in any waveform
EMCCD_gain = 181.749846

Pre_Readout_Time = 0 if do1038Trap else 43.3446849926
pre_readout_frequency = 94.0404898782 #profile 5
pre_readout_power = -5.30880810422 #profile 5

grey_cooling_time=0 #this time plus 6ms is the actual time (due to shutter timings)
grey_cooling_freq=137.972


### Optical Pumping ###

optical_pumping_time = 2.28983292731
optical_pumping_frequency = 98.0541205654
optical_pumping_amp = -0.358305975447
OP_2nd_time = 0
OP_hyperfine_frequency = 278.5 # controlled by freq generator not currently connected to
    computer
OP_hyperfine_power = 0 # controlled by freq gen or waveplates, must be stepped by hand
depumping_894_time = 4 # 2 ms, used for optimzation
single_site_459_OP_time = 0
Op_Hf_extratime = 0.0113537818625 # how long should the repump stay on after OP is off


### Raman ###

raman_459_pulse = 0
raman_detuning=-19.677
raman_pulse_at_freq_2 = 0
ramsey_gap_time = 0

# for raman 459 laser AOM
raman_freq = 143.375
```

```
raman_power = 0

# for variable phase gates using 459 Raman light
phase_site1_pulse_time = 0.015625 # set to make a better Cz based on CNOT phase measurements
phase_site2_pulse_time = 0.0125 # set to make a better Cz based on CNOT phase measurements, was
    0.01550388 set for Grover search


### microwaves ###

microwave_pi_pulse = 0.046980904846724854
microwave_pi_by_2 = 0.024043454856288376
microwave_2pi_pulse = 2*microwave_pi_pulse
microwave_pi_by_3 = microwave_pi_pulse / 3.0
parity_phase=0
microwave_freq = 199.9997627743545 # nominally 200 MHz for 9.192631770 hyperfine resonance
microwave_power = -0.156241917688

# for stark shifted microwaves:
uwave_freq_offset1 = 0 # was -0.04585 for shift into resonance
uwave_freq_offset2 = 0 # was -0.04324 for shift into resonance

uwave_Ramsey_gap_LS_off=0
Ramsey_phase=0
rydA_starkshift_amp=-6.83
CNOT_phase = 5.379881452179586 # for blockade > no blockade, subtract pi for no blockade >
    blockade

microwave_pi2_phase = CNOT_phase

### Rydberg ###

# 459
rydberg_A_459_time_control_pi = 0.000414327482797
rydberg_A_459_time_target_pi = 0.000448575969385
rydberg_A_459_time_site3_pi = 0.000525662201527
rydberg_CP_phase = pi/4

# 2 photon Rabi
rydberg_RFE_pi_time_1 =0.00100422686648# 0.00092152053906 # set for nominal delays. For 1038
    _of_in_Rydberg, use 0.000990595325118+.000050
difference_between_control_pi_1_and_control_pi_2 = 0
rydberg_RFE_2pi_time_2 =0.00180218354061# 0.00186846426297
rydberg_RFE_pi_time_2 = rydberg_RFE_2pi_time_2 / 2
rydberg_RFE_pi_time_3 = 0.000719100549482
second_pulse_time=0 #second pulse pi time 0.00095228

# timing
extra_delay=0 # extra delay in cz gate
ryd_mag_wait = 0.520647359994
no_blockade_wait=5

# These offsets are the sum total of the AOM frequency shifts for the 2-photon rydberg light. #
    swapped
# Set this using a TPS experiment. The 459 switch frequency will change to compensate in the
    dependent variables.
```

```
site_1_RydA_frequency_offset = 302.70488610619867+0.0518940068288 +0.193 +0.186 +0.049 -0.037
    +0.079 -0.1278 -0.001441 -0.0125 # site 36
site_2_RydA_frequency_offset = 302.78459692347343-0.00364065531018 +0.208 +0.2015 +0.048 +0.009
    +0.046 -0.1599 -0.02482 +0.0189 # site 22
site_3_RydA_frequency_offset = 303.9208481236127

# Rydberg B (not currently used)
rydberg_B_1038_time_control = 0
rydberg_B_459_time_control = 0
rydberg_B_459_time_target = 0
RydB_switch_AOM_frequency_1 = 143
RydB_switch_AOM_frequency_2 = 143


### Beam scanners ###

# 1038 scanner 1: best efficiency 147 (centered on ~site 29)
# 1038 scanner 2: best efficiency 160 (centered on ~site 29)

# scanner frequencies for site 1 (36)
scanner_1_frequency_1_459 = 141+1.5
    +0.39+0.222884324146-1+0.493+0.805+0.366+0.0868155168215+0.315212577304-0.7920
scanner_2_frequency_1_459 = 148+1.4 -0.126+0.35808028156+3.22+0.178
    +0.758+0.28+0.229270539601+0.173357167019+0.001113
scanner_1_frequency_1_1038 = 150-3.25-0.877-0.13-0.52+0.426362469435 -0.0551
scanner_2_frequency_1_1038 = 150+1.8-1.266+0.0870343173826 +0.1296

balance_factor1038 = 0 # was -.5 # factor to balance both sites

# scanner frequencies for site 2 (22)
scanner_1_frequency_2_459 = 141+1.5+5
    +0.24+0.180584146561-1+0.361+1.002+0.291+0.0662026289501+0.334102839822-0.8602
scanner_2_frequency_2_459 = 148+1.4 -0.4268+0.461947485083+2.94+0.214
    +0.804+0.264+0.26821292278+0.161219586188-0.03948
scanner_1_frequency_2_1038 = 150+3.25-0.727-0.03-0.15+0.114509224724 -0.1797
scanner_2_frequency_2_1038 = 150-0.93-0.57+0.0708502277652 +0.08771

# scanner frequencies for site 3 (32)
scanner_1_frequency_3_459 = (151.3442542607478 +0.512 -1.25-0.95790709854 +0.9973 +
    156.33873138212303 +0.4 -1.077 -0.1084 )/2 -0.755
scanner_2_frequency_3_459 = 154.96796762492144-0.535582232016 +1.097 +1.82 -0.6392 - 4 -0.25
scanner_1_frequency_3_1038 = 142.9986134303044+0.0939437725381 -0.176 -0.1414 -0.099
scanner_2_frequency_3_1038 = 159.48602642618764+0.116047769885 -0.2694


### timing ###

delay_no_1038_in_Rydberg=.000120 # subtracted from end of 1038 in 1st pulse and added to
    beginning of 1038 in last pulse, so that 1038 is never on in Rydberg by itself

PGC_to_bias_B_field_delay = 2.99383121373
delay_before_blowaway = 0 # was 5

trap_modulation_time = 0 #not currently controlled by computer
trap_release_time = .020 # used for optimization
delay_between_camera_shots = 35 # 31.3 spec'd mnimum wasn't enough
```

```
DDS_profile_delay = .00006
RydA_onoff_delay_time = .000375
RydA_on_delay = .000325
RydA_off_delay = .000400

# delays to line up all the pulses
scanner_delay__extra_time_459 = .0003
scanner_delay__extra_time_1038 = scanner_delay__extra_time_459
Ryd1038_onoff_delay_time = .000650

HSDIO2_delay = .010970
Ryd1038_on_delay = .000156+.00023
Ryd1038_off_delay = -.000218+0.000044

Ryd1038_on_delay2 = 0#.000156+.000262
Ryd1038_off_delay2 =0# -.000238

# added raw Ryd1038_on_delay to make sure scanner is switched before pulse starts
Ryd1038_scanner1_delay = .000110+.000156
Ryd1038_scanner2_delay = .000070+.000156

Ryd459A_on_delay = .000090
Ryd459A_off_delay = .000012

Ryd459A_on_delay2 =0# .000090
Ryd459A_off_delay2 =0# .000012

# add Ryd459A_on_delay to make sure scanner is switched before pulse starts
Ryd459_scanner1_delay = .000350+.000090
Ryd459_scanner2_delay = .000400+.000090
delay_before_switching =.000350+0.000150
# SHG
trap780top_off_delay = .000670+0.0005
trap780top_on_delay = .000260
# TiSapph
trap780bottom_off_delay = .000894+0.0005
trap780bottom_on_delay = .000070
# wait as little time as possible before excitation
delay_before_excitation_pulse = max(HSDIO2_delay,
                                                    Ryd1038_on_delay, Ryd1038_scanner1_delay
                                                        , Ryd1038_scanner2_delay,
                                                    Ryd459A_on_delay, Ryd459_scanner1_delay,
                                                         Ryd459_scanner2_delay,
                                                    trap780top_off_delay,
                                                        trap780bottom_off_delay
                                                    ) if doGates else 5
# time it takes to fully switch sitesRyd
switching_delay = max(Ryd1038_on_delay, Ryd1038_scanner1_delay, Ryd1038_scanner2_delay,
                                                    Ryd459A_on_delay, Ryd459_scanner1_delay,
                                                         Ryd459_scanner2_delay,
                                                    ) if doGates else 5
# buffer to make sure everything is off again
delay_after_excitation_pulse = .000790

_685_time = 0
frequency_685 = 194
power_685 = 0
```

```
close_3D_shutter_time = 10.75
close_HF_shutter_time = 0
close_shutter_time = 10.75
lifetime_delay = 0
T2_delay = 3

slow_noise_eater_laser_time = 1.5 # time for each slow_noise_eater laser measurements
slow_noise_eater_magnetic_time = 3 # time for each magnetic field axis calibration


### magnetic fields ###

B_antihelmholtz = 1.4585356911757419

MOT25 = -0.725105444533
MOT36 = 0.419825949767
MOTb = 0
MOTv = -0.2536193

PGC25 = -0.849223557474
PGC36 = 1.0521398316
PGCb = 0
PGCv = -0.212002706826

PR25 = 0.0137550618104
PR36 = -0.041390028587
PRb = 0
PRv = -0.0107629184279

RO25 = -0.456076059135
RO36 = 0.457737110679
ROb = 0
ROv = -0.262290387664

EXP25 = -0.0285618703994
EXP36 = -0.0187129311724
EXPb = 2.182
EXPv = -0.270314008062

RYD25 = -0.0285618703994
RYD36 = -0.0187129311724
RYDb = 2.182
RYDv = -0.270314008062


### report ###

# sorts a list in place
def inplacesort(list):
    list.sort()
    return list

# quick save report strings to file
def tofile(path,str):
    with open(path,'w') as f:
        f.write(str)
```

```
    return str

#for pretty printing of j,m_j levels
def asHalf(f):
    n=int(f*2)
    if (n%2) == 0:
        return str(int(f))
    else:
        return str(n)+'/2'


### random benchmarking ###
#import random_benchmarking


### gate set tomography ###
#import gate_set_tomography
#GST_scripts, GST_lengths = gate_set_tomography.template_to_HSDIO_scripts(r'E:\AQuA_settings\GST
    \Phase1DataTemplate1a.txt')
```

B AQuA Dependent Variables

```
####### experiment specific #######


####### dependent calculations #######

### MOT ###

hyperfine_time = MOT_loading_time


### PGC ###

PGC_hyperfine_time = PGC_1_time


### Optical Pumping ###

OP_field_time = PGC_to_bias_B_field_delay+optical_pumping_time+Op_Hf_extratime #+
    depumping_894_time

### microwaves ###

microwave_pulse_3pi_by_4 = microwave_pi_pulse*(3/4)
microwave_Raman_time = microwave_pi_pulse

#microwave_phase_pi2_pulse_site1 corresponds to DDS box 2 channel 3 profile 5,
    microwave_phase_pi2_pulse_site1 corresponds to DDS box 2 channel 3 profile 2
microwave_phase_pi2_pulse_site1 = microwave_pi2_phase #over-write this in grover experiments
microwave_phase_pi2_pulse_site2 = microwave_pi2_phase #over-write this in grover experiments

### composite pulses ###

k_c=arcsin(sin(3.1416*microwave_pi_pulse/microwave_2pi_pulse)/2)/6.283*microwave_2pi_pulse
#CORPS1=microwave_2pi_pulse + microwave_pi_pulse/2 - k_c
#CORPS2=microwave_2pi_pulse -2*k_c
```

```
#CORPS3=microwave_pi_pulse/2 - k_c
CORPS1=delay_before_excitation_pulse+7*microwave_pi_by_3
CORPS2=delay_before_excitation_pulse+5*microwave_pi_by_3
CORPS3=delay_before_excitation_pulse+microwave_pi_by_3
CORPS=CORPS1+CORPS2+CORPS3


### Rydberg ###

rydberg_A_459_time_control = rydberg_RFE_pi_time_1
rydberg_A_459_time_control_2 = difference_between_control_pi_1_and_control_pi_2 +
    rydberg_A_459_time_control
rydberg_A_459_time_target = rydberg_RFE_2pi_time_2 #CAUTION: sometimes this is 2pi time!
rydberg_A_459_time_site3 = 2*rydberg_RFE_pi_time_3

#for 459 alignment only
#rydberg_A_459_time_control = .8 * rydberg_A_459_time_control_pi
#rydberg_A_459_time_target = .8 * rydberg_A_459_time_target_pi
#rydberg_A_459_time_site3 = .9 * rydberg_A_459_time_site3_pi

rydberg_A_1038_time_control = rydberg_A_459_time_control
rydberg_A_1038_time_control_2 = rydberg_A_459_time_control_2
rydberg_A_1038_time_target = rydberg_A_459_time_target
rydberg_A_1038_time_site3= rydberg_A_459_time_site3

#take out when doing ground-Rydberg Ramsey experiment
Ryd_ramsey_delay = 2*rydberg_RFE_pi_time_2 + 2*switching_delay

# These frequency offsets are used to calculate switchyard frequency for each site given new
    scanner values after alignments.
# The sum of all the frequency shifts should be constant given that the rydberg level and beam
    powers are not changed

RydA_switch_AOM_frequency_1 = .5 * ( site_1_RydA_frequency_offset
                                                                  -(
                                                                      scanner_1_frequency_1_459
                                                                       -
                                                                      scanner_2_frequency_1_459
                                                                      )
                                                                  -(
                                                                      scanner_1_frequency_1_1038
                                                                       -
                                                                      scanner_2_frequency_1_1038
                                                                      ) )
RydA_switch_AOM_frequency_2 = .5* ( site_2_RydA_frequency_offset
                                                                 -(
                                                                     scanner_1_frequency_2_459
                                                                      -
                                                                     scanner_2_frequency_2_459
                                                                     )
                                                                 -(
                                                                     scanner_1_frequency_2_1038
                                                                      -
                                                                     scanner_2_frequency_2_1038
                                                                     ) )
RydA_switch_AOM_frequency_3 = .5 * ( site_3_RydA_frequency_offset
```

```
                                                                    -(
                                                                        scanner_1_frequency_3_459
                                                                         -
                                                                        scanner_2_frequency_3_459
                                                                        )
                                                                    -(
                                                                        scanner_1_frequency_3_1038
                                                                         -
                                                                        scanner_2_frequency_3_1038
                                                                        ) )
sitetosite_difference = site_1_RydA_frequency_offset - site_2_RydA_frequency_offset


### Analog Output equations ###

# Table of magnetic field settings. Rows are gradient, x25, x36, vert, x14. Columns are:
MOT=0; PGC_1=1; readout=2; _685=3; Exp=4; ryd=5; prereadout=6; gradient_only=7; x1_4only=8;
    x2_5only=9; x3_6only=10; vertical_only=11; OFF=12;

magnetic_fields = np.array(
    [[B_antihelmholtz, 0,      0,        0,      0,        0,         0,       1, 0, 0, 0, 0, 0],
      [MOT25,            PGC25, RO25, 0,      EXP25, RYD25, PR25, 0, 0, 1, 0, 0, 0],
      [MOT36,            PGC36, RO36, -0.55, EXP36, RYD36, PR36, 0, 0, 0, 1, 0, 0],
      [MOTv,               PGCv,  ROv,  -1.55, EXPv,  RYDv,  PRv,  0,  0, 0, 0, 1, 0],
      [MOTb,              PGCb,  ROb,  0,      EXPb,  RYDb,  PRb,  0, 1, 0, 0, 0, 0]])

# coil driver calibrations. Drivers: CoilDriver.v2h.10, v2h.3, v2h.4, v2g.2, v2g.1
offset = np.array([-0.00144668, -0.0000453172, +0.000748044, -.00129657, -.00261032])
scale = np.array([-0.995666, -0.996497, -0.999946, -0.994236, -1.00318])
magnetic_fields = ((magnetic_fields.T + offset)/scale).T


### Report calculations below ###

# mot/readout parameter input
mot_beam_waist_1 = 2.5 # mm
mot_beam_waist_2 = 2.5 # mm
mot_beam_waist_3 = 1.54 # mm
readout_beam_power_1 = 560 # uW
readout_beam_power_2 = 415 # uW
readout_beam_power_3 = 205 # uW
readout_detuning = 18 # MHz
numerical_aperture_of_jenoptiq = .4
transmission_efficiency = .8
quantum_efficiency = .7
conversion_factor_for_camera = 5.8 # electrons/count
analog_gain = 4
EM_sensitivity = int(EMCCD_gain) # from independent variable
EM_gain = 10**(((log(1000)-log(4))/250)*EM_sensitivity+log(4))
average_background_pixel_value = 10914*(Readout_time/.100)
roi_area = 9
imaging_system_mag = 25.5
camera_pixel_size = 16 # um

# array parameter input
number_of_array_spots = 64 # n
array_trap_spacing = 3.8 # d_microns
```

```
array_beam_waist = 1.73 # w0_microns
power_out_of_the_fiber_780 = 5 # W
transmission_to_atoms_780 = .45

# excitation beam parameter input
scanner_transmission_459 = .132
scanner_transmission_1038 = .25

horizontal_waist_459 = 3e-6
vertical_waist_459 = 3e-6
horizontal_waist_1038 = 3.2e-6
vertical_waist_1038 = 4.3e-6

total_raman_power_at_fiber = 263.8e-6
raman_detuning = 20e9 #Hz

Rydberg_A_459_power_at_fiber =303e-6
Rydberg_B_459_power_at_fiber = 120e-6
Rydberg_1038_power_at_fiber = 10e-3
Rydberg_n = 82
Rydberg_l = 0 #D level
Rydberg_j = 1/2
Rydberg_m = 1/2
Rydberg_detuning = .658e9 #Hz relative to f = (4 -> 4')

# mot/readout calculations
readout_detuning = readout_detuning*2*pi*1e6
readout_beam_intensity_1 = 2*readout_beam_power_1*1e-6/(pi*mot_beam_waist_1**2*1e-6)
readout_beam_intensity_2 = 2*readout_beam_power_2*1e-6/(pi*mot_beam_waist_2**2*1e-6)
readout_beam_intensity_3 = 2*readout_beam_power_3*1e-6/(pi*mot_beam_waist_3**2*1e-6)
total_intensity = 2*readout_beam_intensity_1+2*readout_beam_intensity_2+2*
    readout_beam_intensity_3
total_saturation_parameter = total_intensity/saturation_intensity
fractional_solid_angle = .5*(1-sqrt(1-numerical_aperture_of_jenoptiq**2))
scattering_rate = (gamma_6p3_2/2)*total_saturation_parameter/(1+(4*readout_detuning**2)/(
    gamma_6p3_2**2)+total_saturation_parameter)
detected_scattering_rate_for_single_atom = scattering_rate*fractional_solid_angle*
    transmission_efficiency*quantum_efficiency
detected_photon_number_during_exposure_time = detected_scattering_rate_for_single_atom*
    Readout_time*(1e-3)
expected_output_signal_dark_signal = detected_photon_number_during_exposure_time*analog_gain*
    EM_gain*quantum_efficiency/conversion_factor_for_camera
expected_output_signal = expected_output_signal_dark_signal+average_background_pixel_value*
    roi_area
pixel_size_after_mag = camera_pixel_size/imaging_system_mag #um

# array calculations
lattice_period = array_trap_spacing/pixel_size_after_mag #in pixels
array_ratio = array_trap_spacing/array_beam_waist #s=d/w
total_power_at_atoms = power_out_of_the_fiber_780*transmission_to_atoms_780 #W
P0 = (2*total_power_at_atoms/number_of_array_spots)/(pi*array_beam_waist**2e-12) #
    peak_intensity_in_one_spot
trap_depth = -(0.001*2*pi*alpha_780_cgs/(kB_SI*c0))*P0*2*exp(-array_ratio**2/2)*(1-2*exp(-
    array_ratio**2/2)) #mK

# raman calculations
```

```
Raman_Rabi_freq = total_raman_power_at_fiber/2*scanner_transmission_459/(2*pi*
    horizontal_waist_459*vertical_waist_459)*((-1.305e9)/(raman_detuning-
    hyperfine_splitting_7p1_2)-7.83e8/raman_detuning) #Hz
Raman_Stark_shift = total_raman_power_at_fiber/2*scanner_transmission_459/(2*pi*
    horizontal_waist_459*vertical_waist_459)*(4.1e9/(2*pi*(raman_detuning-
    hyperfine_splitting_7p1_2-hyperfine_splitting_6s1_2))-4.1e9/(2*pi*(raman_detuning-
    hyperfine_splitting_7p1_2+hyperfine_splitting_6s1_2)+2.46e9/(2*pi*(raman_detuning-
    hyperfine_splitting_6s1_2))-2.46e9/(2*pi*(raman_detuning+hyperfine_splitting_6s1_2)))) #Hz
Rydberg_A_flopping_to_nD3_2 = 1/(2*pi*Rydberg_n**(3/2))*sqrt(Rydberg_A_459_power_at_fiber*
    scanner_transmission_1038*Rydberg_1038_power_at_fiber*scanner_transmission_1038/(
    horizontal_waist_459*vertical_waist_459*horizontal_waist_1038*vertical_waist_1038))*(9.692
    e10/(Rydberg_detuning)+5.815e10/(Rydberg_detuning+hyperfine_splitting_7p1_2)) #Hz
Rydberg_B_flopping_to_nD3_2 = 1/(2*pi*Rydberg_n**(3/2))*sqrt(Rydberg_B_459_power_at_fiber*
    scanner_transmission_1038*Rydberg_1038_power_at_fiber*scanner_transmission_1038/(
    horizontal_waist_459*vertical_waist_459*horizontal_waist_1038*vertical_waist_1038))*(9.692
    e10/(Rydberg_detuning)+5.815e10/(Rydberg_detuning+hyperfine_splitting_7p1_2)) #Hz
Raman_459_power_out_of_fiber = total_raman_power_at_fiber*1e3 #mW
Raman_459_transmission_of_scanner = scanner_transmission_459
Raman_459_power_at_atoms = Raman_459_power_out_of_fiber*Raman_459_transmission_of_scanner #mW
Raman_459_detuning = raman_detuning*1e-9 #GHz
Raman_459_waist_at_atoms = sqrt(horizontal_waist_459*vertical_waist_459)*1E6 #um
Raman_459_7p1_2_Delta_f3 = -.2123 #GHz
Raman_459_7p1_2_Delta_f4 = .1651 #GHz
Raman_459_intermediate = 1/(Raman_459_detuning-Raman_459_7p1_2_Delta_f3)+(5/3)/(
    Raman_459_detuning-Raman_459_7p1_2_Delta_f4)
Raman_459_Rabi_Omega_over_2pi = 93.3*Raman_459_intermediate*Raman_459_power_at_atoms/
    Raman_459_waist_at_atoms**2 #MHz
Raman_459_gamma_p = 1/150 #GHz
Raman_459_omega_q = 9.192 #GHz
Raman_459_Pse_in_pi_pulse = Raman_459_gamma_p/4*(1/abs(Raman_459_intermediate))*(2/(
    Raman_459_detuning-Raman_459_7p1_2_Delta_f3)**2+1/(Raman_459_detuning-
    Raman_459_7p1_2_Delta_f3+Raman_459_omega_q)**2+1/(Raman_459_detuning-
    Raman_459_7p1_2_Delta_f3-Raman_459_omega_q)**2+(10/3)/(Raman_459_detuning-
    Raman_459_7p1_2_Delta_f4)**2+(5/3)/(Raman_459_detuning-Raman_459_7p1_2_Delta_f4+
    Raman_459_omega_q)**2+(5/3)/(Raman_459_detuning-Raman_459_7p1_2_Delta_f4-Raman_459_omega_q)
    **2)
Raman_459_differential_stark_shift = Raman_459_Rabi_Omega_over_2pi*(Raman_459_detuning/32)
    *((5/3)/(Raman_459_detuning-Raman_459_7p1_2_Delta_f4+Raman_459_omega_q)+1/(
    Raman_459_detuning-Raman_459_7p1_2_Delta_f3+Raman_459_omega_q)-(5/3)/(Raman_459_detuning-
    Raman_459_7p1_2_Delta_f4-Raman_459_omega_q)-1/(Raman_459_detuning-Raman_459_7p1_2_Delta_f3-
    Raman_459_omega_q)) #MHz

# Rydberg calculations
Rydberg_459_A_power_at_atoms = Rydberg_A_459_power_at_fiber*scanner_transmission_459*(1e3) #mW
Rydberg_1038_power_at_atoms = Rydberg_1038_power_at_fiber*scanner_transmission_1038*(1e3) #mW
Rydberg_459_beam_waist = sqrt(horizontal_waist_459*vertical_waist_459)*(1e6)
Rydberg_1038_beam_waist = sqrt(horizontal_waist_1038*vertical_waist_1038)*(1e6)
Rydberg_detuning_3p = -212.3*(2*pi) #MHz
Rydberg_detuning_4p = 165.1*(2*pi) #MHz
Rydberg_detuning = Rydberg_detuning*1e-9 +.165#GHz F=4 to 7P center of gravity
Rydberg_detuning_over_2pi = Rydberg_detuning*(2*pi)*1e3 #MHz

if Rydberg_l==0: #nS
    Rydberg_Rabi_Frequency_over_2pi = 20600*(sqrt(Rydberg_459_A_power_at_atoms*
        Rydberg_1038_power_at_atoms)/(Rydberg_n**(3/2)*Rydberg_459_beam_waist*
        Rydberg_1038_beam_waist*Rydberg_detuning))*(1-(5/8)*(Rydberg_detuning_3p/
        Rydberg_detuning_over_2pi)-(3/8)*(Rydberg_detuning_4p/Rydberg_detuning_over_2pi))/((1-(
```

```
        Rydberg_detuning_3p/Rydberg_detuning_over_2pi))*(1-(Rydberg_detuning_4p/
        Rydberg_detuning_over_2pi))) #MHz
elif Rydberg_l==2: #nD
    Rydberg_Rabi_Frequency_over_general = 240600*(sqrt(Rydberg_459_A_power_at_atoms*
        Rydberg_1038_power_at_atoms)/(Rydberg_n**(3/2)*Rydberg_459_beam_waist*
        Rydberg_1038_beam_waist*Rydberg_detuning)) # MHz
    angluar_factor_m3over2=1/(2*sqrt(6))
    Rydberg_Rabi_Frequency_over_2pi=Rydberg_Rabi_Frequency_over_general*angluar_factor_m3over2

Rydberg_AC_stark_shift = (160200*(Rydberg_1038_power_at_atoms/(Rydberg_n**3*
    Rydberg_1038_beam_waist**2))-93.47*(Rydberg_459_A_power_at_atoms/Rydberg_459_beam_waist**2))
    *(1/Rydberg_detuning)*((1/(1-Rydberg_detuning_3p/Rydberg_detuning_over_2pi))+((5/3)/(1-
    Rydberg_detuning_4p/Rydberg_detuning_over_2pi)))
Rydberg_sponteneous_emission_rate = (.43*(sqrt(Rydberg_1038_power_at_atoms)*
    Rydberg_459_beam_waist/(Rydberg_n**(3/2)*sqrt(Rydberg_459_A_power_at_atoms)*
    Rydberg_1038_beam_waist))+(2.5e-4)*(sqrt(Rydberg_459_A_power_at_atoms)*
    Rydberg_1038_beam_waist/(Rydberg_n**(3/2)*sqrt(Rydberg_1038_power_at_atoms)*
    Rydberg_459_beam_waist)))*(1/Rydberg_detuning)*(1-2*((5/8)*(Rydberg_detuning_3p/
    Rydberg_detuning_over_2pi)+(3/8)*(Rydberg_detuning_4p/Rydberg_detuning_over_2pi))+(5/8)*(
    Rydberg_detuning_3p/Rydberg_detuning_over_2pi)**2+(3/8)*(Rydberg_detuning_4p/
    Rydberg_detuning_over_2pi)**2)/((1-Rydberg_detuning_3p/Rydberg_detuning_over_2pi)*(1-
    Rydberg_detuning_4p/Rydberg_detuning_over_2pi)*(1-(5/8)*(Rydberg_detuning_3p/
    Rydberg_detuning_over_2pi)-(3/8)*(Rydberg_detuning_4p/Rydberg_detuning_over_2pi)))
Rydberg_blue_power_needed_to_match_red_Rabi = (1/scanner_transmission_459)*(
    Rydberg_1038_power_at_atoms*Rydberg_459_beam_waist**2)/(.00058*Rydberg_1038_beam_waist**2*
    Rydberg_n**3)

power_894 = .450 #uW
OP_repumper = 8.5 #uW
```

C AQuA Functional Waveforms

```
"""These are the waveform functions for the AQuA project.
They define the operation of the HSDIO, AO and DAQmxDO outputs.
Use the functions HSDIO(time, channel, state), DO(time, channel, state), AO(time, channel,
    voltage) and label(time, text).
The user should ensure that all waveform functions take in the start time as the first parameter
    ,
and return the end time.
"""

# reset the time
t = 0

HSDIO = experiment.LabView.HSDIO.add_transition
AO = experiment.LabView.AnalogOutput.add_transition
DO = experiment.LabView.DAQmxDO.add_transition
label = experiment.functional_waveforms_graph.label

class DDS(object):
    """This class represents a single DDS channel controlled by one or more HSDIO channels.
    This class does NOT communicate directly with the DDS box, that is taken care of in the DDS.
        py file.
    This class only manipulates the HSDIO channels which switch the current HSDIO profile.
    It takes care of grey coding the profile changes.
    This class only works properly if all calls to a particular instance are done sequentially,
        but that is the expected situation for a single DDS channel."""
```

```
def __init__(self, t, channels, profiles=None):
    """Create a new DDS channel.
    channels: a list of the HSDIO channels corresponding to the DDS bits. e.g. (0, 1, 18)
    profile: a dict using profile names as keys to look up the bit settings. e.g. {"MOT":
        (0,0,0), "OFF": (1,0,0)}
    """

    self.channels = channels
    self.profiles = profiles

    # keep track of the state
    self.bits = [False, False, False]
    # keep track of the last time a bit was changed, to see if we need to add a DDS delay
    # assume that a bit was changed immediately before this, to be conservative
    self.last_change = t

def initialize(self, t, new_bits):
    """Sets the state. Unlike set(), this function sets every bit, regardless of its current
        state."""
    # set the state in the HSDIO, adding a delay between each bit setting if necessary
    for channel, new_bit in zip(self.channels, new_bits):
        t = self.delay(t)
        HSDIO(t, channel, new_bit)
    # keep track that the bits have all been set
    self.bits = new_bits
    return t

def delay(self, t):
    """Add a DDS delay if it is needed, and update the last_change parameter."""

    if t <= (self.last_change + DDS_profile_delay):
        t += DDS_profile_delay
    # update the last_change parameter
    self.last_change = t
    return t

def set(self, t, new_bits):
    """Sets the state. Only bits that need to be flipped are flipped. Grey coding is
        automatic.
    new_bits: the state we want to set"""

    # go through the bits one at a time
    for channel, old_bit, new_bit in zip(self.channels, self.bits, new_bits):
        # check to see if the bit needs to be changed
        if old_bit != new_bits:
            # delay if we recently changed another bit
            t = self.delay(t)
            HSDIO(t, channel, new_bit)
    # keep track that the bits have all been set
    self.bits = new_bits
    return t

def profile(self, t, profile):
    """Switch to a specific profile by looking up the name in the stored dict."""
    return self.set(t, self.profiles[profile])
```

```
### set up the DDS channels to use the DDS class for grey coding ###
# Alias the the profile() method of each of these instances, for convenience.
MOT = DDS(t, (0, 1, 18), {'MOT':(0,0,0), 'PGC in MOT':(1,0,0), 'off':(0,1,0), 'readout':(1,1,0),
     'light assisted collisions':(1,0,1), 'off 2':(0,1,1), 'PGC in traps':(1,1,1)}).profile
repump = DDS(t, (2,), {'on':(0,), 'off':(1,)}).profile
SHG780 = DDS(t, (12,), {'on':(1 if all780off else 0,), 'off':(1,)}).profile
Verdi780 = DDS(t, (3,), {'on':(0 if all780off else 1,), 'off':(0,)}).profile
uwave_DDS = DDS(t, (14, 10, 28), {'global':(0,0,0), 'site 1 0 phase':(1,0,0), 'site 2 CNOT phase
    ':(0,1,0), 'global parity phase':(1,1,0), 'site 2 0 phase':(0,0,1), 'site 1 CNOT phase
    ':(1,0,1)}).profile
scanner459_1 = DDS(t, (16, 17), {'off':(0,0), 'site 1':(1,0), 'site 2':(0,1), 'site 3':(1,1)}).
    profile
scanner459_2 = DDS(t, (19, 20), {'off':(0,0), 'site 1':(1,0), 'site 2':(0,1), 'site 3':(1,1)}).
    profile
scanner1038_1 = DDS(t, (22,), {'site 1':(0,), 'site 2':(1,)}).profile
scanner1038_2 = DDS(t, (23,), {'site 1':(0,), 'site 2':(1,)}).profile
Rydberg459A = DDS(t, (24, 31), {'off':(0,0), 'site 1':(1,0), 'site 2':(0,1), 'site 3':(1,1)}).
    profile
Rydberg1038 = DDS(t, (21,29), {'off':(0,0), 'on':(1,0), 'low power':(0,1)}).profile

class switch(object):
    """A single HSDIO channel that controls a switch. Unlike DDS(), this does not use any grey
        coding or delays."""

    def __init__(self, channel, profiles=None):
        """
        channel: the HSDIO channel that controls this device
        profiles: a dict of profile settings, e.g {'on':1, 'off':2}
        """
        self.channel = channel
        self.profiles = profiles

    def profile(self, t, profile):
        """Set the HSDIO channel to the requested state"""
        HSDIO(t, self.channel, self.profiles[profile])
        return t

### set up the switches that do not need grey coding ###
# Alias the the profile() method of each of these instances, for convenience.
MOT2D_shutter = switch(4, {'on':0, 'off':1}).profile
MOT3D_shutter = switch(9, {'on':0, 'off':1}).profile
repump_shutter = switch(25, {'on':0, 'off':1}).profile
OP = switch(26, {'on':1, 'off':0}).profile
OP_repump = switch(30, {'on':1, 'off':0}).profile
uwave_switch = switch(13, {'on':1, 'off':0}).profile
Raman459 = switch(15, {'on':1, 'off':0}).profile
blowaway = switch(11, {'on':1, 'off':0}).profile
slow_noise_eater_trigger2 = switch(27, {'on':1, 'off':0}).profile

### Create a special Hamamatsu class so we can keep track of when the last shot was
class Hamamatsu_class(switch):
    """A special case of switch that also keeps track of when the last shot was."""

    def __init__(self, t):
        super(Hamamatsu_class, self).__init__(5, {'open':1, 'closed':0})
        self.last_shot = t
```

```
Hamamatsu = Hamamatsu_class(t)

#### define the component waveforms that can be mixed and matched to make an experiment ###

def set_magnetic_fields(t, profile):
    """Switch to the magnetic fields of choice, defined by the magnetic fields matrix."""

    # Table of magnetic field settings. Rows are gradient, x25, x36, vert, x14. Columns are:
    profiles = {'MOT':0, 'PGC in MOT':1, 'readout':2, '685':3, 'optical pumping':4, 'Rydberg':5,
         'light assisted collisions':6,
               'gradient only':7, 'x1_4 only':8, 'x2_5 only':9, 'x3_6 only':10, 'vertical only
                   ':11, 'off':12}

    # for each AO channel (rows of the magnetic field matrix)
    for channel, fields in enumerate(magnetic_fields):
        # switch to the field defined by the column of the magnetic fields matrix selected by '
            profile'
        AO(t, channel, fields[profiles[profile]])

    return t

def DCNE_trigger_1(t):
    """ The once-per-measurement trigger for the DC Noise Eaters. Goes low for 1 ms."""
    DO(t, 0, 0)
    t += 1
    DO(t, 0, 1)

def MOT_loading(t):
    """Load atoms from the 2D vapor beam"""

    label(t, 'MOT loading')

    # turn on the MOT, repump and traps, everything else is off
    if background:
        t1 = MOT(t, 'off')
        t2 = repump(t, 'off')
        t3 = MOT2D_shutter(t, 'off')
    else:
        t1 = MOT(t, 'MOT')
        t2 = repump(t, 'on')
        t3 = MOT2D_shutter(t, 'on')
    t4 = MOT3D_shutter(t, 'on')
    t5 = SHG780(t, 'on')
    t6 = Verdi780(t, 'on')

    t = max(t1, t2, t3, t4, t5, t6)


    # turn off the 780 traps

    t1 = SHG780(t + trap_on_time_during_loading, 'off')
    t2 = Verdi780(t + trap_on_time_during_loading, 'off')

    # turn off the 2D MOT
    t3 = MOT2D_shutter(t + MOT_2D_time, 'off')

    # end sequence
```

```
    t4 = t + MOT_loading_time

    t = max(t1, t2, t3, t4)

    return t

def PGC_in_MOT(t):
    """Polarization gradient cooling in the MOT."""

    label(t, 'PGC in MOT')

    # calculate when the end of the sequence will come
    t1 = t+PGC_1_time

    # switch to PGC phase
    if not background:
        t = MOT(t, 'PGC in MOT')

    # turn on the traps
    t += trap_780_delay_for_PGC_1
    t_after_SHG = SHG780(t, 'on')
    t_after_Verdi = Verdi780(t, 'on')
    t = max(t_after_SHG, t_after_Verdi)

    # end sequence
    t = max(t, t1)
    return t

def MOT_drop(t):
    """After single atom traps are on, let the rest of the MOT fall away."""

    label(t, 'MOT drop')

    t1 = MOT(t, 'off')
    t2 = repump(t, 'off')
    t = max(t1,t2)

    # let the MOT drop away
    t += MOT_drop_time

    return t

def light_assisted_collisions(t):
    """Time to allow light assisted collisions eliminate the occurrence of more than 1 atom per
        trap."""

    label(t, 'pre-readout')

    if not background:
        t1 = MOT(t, 'light assisted collisions')
        t2 = repump(t, 'on')
        t = max(t1, t2)

    # wait for light assisted collisions to eliminate twos, threes, etc, from the traps.
    t += Pre_Readout_Time

    return t
```

```
def PGC_in_traps(t):
    """Polarization gradient cooling in the single atom traps."""

    label(t, 'PGC in traps')

    if not background:
        t1 = MOT(t, 'PGC in traps')
        t2 = repump(t, 'on')
        t = max(t1, t2)

    # wait for atoms to cool
    t += PGC_2_time

    return t

def readout(t):
    """Take a picture."""

    label(t, 'readout')

    MOT(t, 'readout')
    repump(t, 'on')
    Hamamatsu.profile(t, 'open')

    # leave the shutter open for the exposure time
    t += Readout_time
    Hamamatsu.last_camera_shot = t
    Hamamatsu.profile(t, 'closed')

    return t

def close_shutters(t):
    """Close the MOT shutters to prevent state mixing."""

    label(t, 'close shutters')

    # start the shutters closing
    MOT(t, 'off')
    MOT3D_shutter(t, 'off')
    repump(t, 'off')
    repump_shutter(t, 'off')

    # wait until they are fully closed
    t += close_shutter_time

    return t

def open_3D_shutters(t):
    """Open the MOT shutters again so we can take a picture."""

    label(t, 'open shutters')

    # start the shutters opening
    MOT3D_shutter(t, 'on')
    repump_shutter(t, 'on')
```

```
    # wait until they are fully open (this could use it's own time, not necessarily the same as
        the close_shutter_time)
    t += close_shutter_time

    return t

def OP_magnetic_fields(t):
    """Turn on a bias field and wait until the field stabilizes."""

    t = set_magnetic_fields(t, 'optical pumping')

    # wait until the fields are fully switched
    t += PGC_to_bias_B_field_delay

    return t

def optical_pumping(t):
    """Shuffle atoms until they are in the F=4, mF=0 dark state. The OP beam is 894 nm linearly
        polarized F=4 to 4'."""

    label(t, 'optical pumping')

    # turn on the OP and OP repumper
    OP(t, 'on')
    OP_repump(t, 'on')

    # turn off the OP and OP repumper
    t += optical_pumping_time
    OP(t, 'off')
    OP_repump(t, 'off')

    return t

def OP_depump(t):
    """Use just the OP beam without repumping.
    Since the F=4, mF=0 state is dark to this, the transference to F=3 can be used to measure
        how good the previous
    optical pumping was."""

    label(t, 'OP depump')

    # turn on the OP, with no OP repumper
    OP(t, 'on')
    OP_repump(t, 'off')

    # turn off the OP
    t += depumping_894_time
    OP(t, 'off')

    return t

def Rydberg_magnetic_fields(t):
    """Set the magnetic shims to the RYD settings and wait until they stabilize."""

    t = set_magnetic_fields(t, 'Rydberg')

    # wait until the fields are fully switched
```

```
    t += ryd_mag_wait

    return t

def no_op(t):
    """Do nothing for zero time."""
    return t

def uwave_global(t, duration):
    """Perform a global X rotation using microwaves.
    duration: the length of the pulse"""

    # switch the microwave DDS profile
    t = uwave_DDS(t, 'global')

    # turn on microwave pulse
    uwave_switch(t, 'on')

    # turn off microwave pulse
    t += duration
    t = uwave_switch(t, 'off')

    return t

def uwave_pi_global(t):
    """Perform a global X gate (X pi rotation"""
    label(t, 'uwave pi global')
    t = uwave_global(t, microwave_pi_pulse)
    return t

def uwave_pi_by_2_global(t):
    """Perform a global X pi/2 rotation."""
    label(t, 'uwave pi/2 global')
    t = uwave(t, microwave_pi_by_2)
    return t

def uwave_wStark_site1_Raman(t, duration):
    """Perform an X rotation on all sites EXCEPT site 1.
    This assumes the microwaves are tuned to resonance, so the 459 'Raman' light will detune the
        microwaves.
    """

    # switch the microwave DDS profile
    t1 = uwave_DDS(t, 'site 1 0 phase')
    # switch the scanners to site 1
    t2 = scanner459_1(t, 'site 1')
    t3 = scanner459_2(t, 'site 1')
    # wait for scanners to switch
    t2 += Ryd459_scanner1_delay
    t3 += Ryd459_scanner2_delay
    t = max(t1, t2, t3)

    # turn on microwaves and 459 light
    uwave_switch(t, 'on')
    Raman459(t, 'on')

    # turn off pulse
```

```
    t += microwave_pi_pulse
    t = uwave_switch(t, 'off')

    # Don't turn 459 scanners to off here. Save time by only doing it after the last pulse.

    return t

def uwave_wStark_site1_Pi_Raman(t):
    """An X gate on all sites EXCEPT site 1"""
    label(t, 'uwave pi site ~1')
    t = uwave_wStark_site1_Raman(t, microwave_pi_pulse)
    return t

def uwave_wStark_site1_Pi2_Raman(t):
    """An X pi/2 rotation on all sites EXCEPT site 1"""
    label(t, 'uwave pi/2 site ~1')
    t = uwave_wStark_site1_Raman(t, microwave_pi_by_2)
    return t

def uwave_wStark_site2_Raman(t, duration):
    """Perform an X rotation on all sites EXCEPT site 2.
    This assumes the microwaves are tuned to resonance, so the 459 'Raman' light will detune the
        microwaves.
    duration: defines the length of the pulse
    """

    # switch the microwave DDS profile
    t1 = uwave_DDS(t, 'site 2 0 phase')
    # switch the scanners to site 2
    t2 = scanner459_1(t, 'site 2')
    t3 = scanner459_2(t, 'site 2')
    # wait for scanners to switch
    t2 += Ryd459_scanner1_delay
    t3 += Ryd459_scanner2_delay
    t = max(t1, t2, t3)

    # turn on microwaves and 459 light
    uwave_switch(t, 'on')
    Raman459(t, 'on')

    # turn off pulse
    t += duration
    t = uwave_switch(t, 'off')

    # Don't turn 459 scanners to off here. Save time by only doing it after the last pulse.

    return t

def uwave_wStark_site2_Pi_Raman(t):
    """X gate on all sites EXCEPT site 2"""
    label(t, 'uwave pi site ~2')
    t = uwave_wStark_site2_Raman(t, microwave_pi_pulse)
    return t

def uwave_wStark_site2_Pi2_Raman(t):
    """X gate on all sites EXCEPT site 2"""
    label(t, 'uwave pi/2 site ~2')
```

```
    t = uwave_wStark_site2_Raman(t, microwave_pi_by_2)
    return t

def uwave_wStark_site2_Raman_phase(t, duration):
    """Perform an X rotation on all sites EXCEPT site 2 with a phase offset.
    This assumes the microwaves are tuned to resonance, so the 459 'Raman' light will detune the
        microwaves.
    duration: defines the length of the pulse
    """

    # switch the microwave DDS profile
    t1 = uwave_DDS(t, 'site 2 CNOT phase')
    # switch the scanners to site 2
    t2 = scanner459_1(t, 'site 2')
    t3 = scanner459_2(t, 'site 2')
    # wait for scanners to switch
    t2 += Ryd459_scanner1_delay
    t3 += Ryd459_scanner2_delay
    t = max(t1, t2, t3)

    # turn on microwaves and 459 light
    uwave_switch(t, 'on')
    Raman459(t, 'on')

    # turn off pulse
    t += duration
    t = uwave_switch(t, 'off')

    # Don't turn 459 scanners to off here. Save time by only doing it after the last pulse.

    return t

def uwave_wStark_site2_Pi2_Raman_phase(t):
    label(t, 'uwave pi/2 site ~2 phase')
    t = uwave_wStark_site2_Raman_phase(t, microwave_pi_by_2)
    return t

def scanners459_off(t):
    """Turn the 459 scanners off. This function is used so we don't have to turn the scanners
        off after the 1st
    microwave pulse in the Cz waveform."""

    # scanners back to off
    t1 = scanner459_1(t, 'off')
    t2 = scanner459_2(t, 'off')
    # wait for scanners to switch
    t1 += Ryd459_scanner1_delay
    t2 += Ryd459_scanner2_delay

    return t

def do_blowaway(t):
    """Remove any atoms in F=4."""

    label(t, 'blowaway')

    # do a blowaway pulse
```

```
    t = blowaway(t, 'on')
    t += blow_away_time
    t = blowaway(t, 'off')

    return t

def camera_delay(t):
    """Wait until 31 ms after the previous camera shot."""

    label(t, 'camera delay')

    # If we have already waited 31 ms doing other operations, then just proceed.
    # Otherwise, wait until 31 ms after the last shot

    return max(t, Hamamatsu.last_shot + delay_between_camera_shots)

def slow_noise_eater(t):
    """Turn on the lasers one by one, so we can get a reading of their power."""

    label(t, 'noise eater')

    # turn everything off
    t = max(MOT(t, 'off'),
           repump(t, 'off'),
           Verdi780(t, 'off'),
           SHG780(t, 'off'),
           scanner459_1(t, 'off'),
           scanner459_2(t, 'off'),
           scanner1038_1(t, 'site 1'),
           scanner1038_2(t, 'site 1')
           )

    # each step will be 2 ms
    dt = slow_noise_eater_laser_time
    dt2 = 0.5 # off for 0.5 ms
    dt3 = 0.005 # noise eater trigger delay, to ensure laser has turned on

    # 'Raman' 459
    Raman459(t, 'on')
    slow_noise_eater_trigger2(t+dt3, 'on')
    t += dt
    Raman459(t, 'off')
    slow_noise_eater_trigger2(t, 'off')
    t += dt2

    # Rydberg 459A
    Rydberg459A(t, 'site 1')
    slow_noise_eater_trigger2(t+dt3, 'on')
    t += dt
    Rydberg459A(t, 'off')
    slow_noise_eater_trigger2(t, 'off')
    # turn the 459 scanner off now
    scanner459_1(t, 'off')
    scanner459_2(t, 'off')
    t += dt2

    # Rydberg 1038
```

```
Rydberg1038(t, 'on')
slow_noise_eater_trigger2(t+dt3, 'on')
t += dt
Rydberg1038(t, 'off')
slow_noise_eater_trigger2(t, 'off')
t += dt2

# MOT
MOT(t,'MOT')
t += dt
MOT(t, 'off')
t += dt2

# repump
repump(t, 'on')
t += dt
repump(t, 'off')
t += dt2

# OP
OP(t, 'on')
t += dt
OP(t, 'off')
t += dt2

# OP repump
OP_repump(t, 'on')
t += dt
OP_repump(t, 'off')
t += dt2

# blowaway
blowaway(t, 'on')
t += dt
blowaway(t, 'off')
t += dt2

# 780 SHG
SHG780(t, 'on')
t += dt
SHG780(t, 'off')
t += dt2

# 780 Verdi TiSapph
Verdi780(t, 'on')
t += dt
Verdi780(t, 'off')
t += dt2

# return to MOT loading settings
if not background:
    MOT(t, 'MOT')
    repump(t, 'on')
    MOT2D_shutter(t, 'on')
    MOT3D_shutter(t, 'on')
SHG780(t, 'on')
Verdi780(t, 'on')
```

```
    return t

def magnetic_field_monitor(t):
    """Turn on each coil one by one, so we can get a reading of their noise from shot to shot
        ."""

    label(t, 'magnetic field monitor')

    #each step will be 3 ms
    dt = slow_noise_eater_magnetic_time

    set_magnetic_fields(t, 'gradient only')
    t += dt
    set_magnetic_fields(t, 'x1_4 only')
    t += dt
    set_magnetic_fields(t, 'x2_5 only')
    t += dt
    set_magnetic_fields(t, 'x3_6 only')
    t += dt
    set_magnetic_fields(t, 'vertical only')
    t += dt
    set_magnetic_fields(t, 'off')
    t += dt

    return t

def trap_drop(t):
    """Turn off the 780 nm traps for a short period of time. Usually to check atom temperature
        ."""

    label(t, 'trap drop')

    # traps off
    SHG780(t, 'off')
    Verdi780(t, 'off')

    # wait a short time with the traps off
    t += trap_release_time

    # turn the traps back on
    SHG780(t, 'on')
    Verdi780(t, 'on')

    return t

def open_2D_shutter(t):
    """Reopen the 2D MOT shutter."""

    label(t, 'open 2D shutter')

    # turn on the MOT, repump and traps, everything else is off
    if background:
        MOT2D_shutter(t, 'off')
    else:
        MOT2D_shutter(t, 'on')
```

```
    return t


def idle(t):
    """Turn on the MOT to start the loading, and turn on the 780 to stabilize the temperature
        between measurements.
    These settings will persist until the next measurement starts."""

    label(t, 'idle')

    # turn on the MOT, repump and traps, everything else is off
    if background:
        MOT(t, 'off')
        repump(t, 'off')
        MOT2D_shutter(t, 'off')
    else:
        MOT(t, 'MOT')
        repump(t, 'on')
        MOT2D_shutter(t, 'on')
    MOT3D_shutter(t, 'on')
    SHG780(t, 'on')
    Verdi780(t, 'on')

    # if you want to turn a specific channel on or off during idle, just do this here:
    #HSDIO(t, 3, 1) # e.g. turn on channel 3

    return t

def before_experiment(t):
    """All the steps to get atoms initialized for an experiment."""
    DCNE_trigger_1(t)
    t = set_magnetic_fields(t, 'MOT')
    t = MOT_loading(t)
    t = set_magnetic_fields(t, 'PGC in MOT')
    t = PGC_in_MOT(t)
    t = set_magnetic_fields(t, 'light assisted collisions')
    t = MOT_drop(t)
    t = light_assisted_collisions(t)
    t = set_magnetic_fields(t, 'readout')
    t = PGC_in_traps(t)
    t = readout(t)
    t = PGC_in_traps(t)
    t = close_shutters(t)
    t = OP_magnetic_fields(t)
    t = optical_pumping(t)
    HSDIO(t,34,1) # oscilloscope trigger on
    return t

def after_experiment(t):
    """All the steps to readout after an experiment.
    Note: Does not include blowaway."""

    HSDIO(t,34,0) # oscilloscope trigger off
    t = set_magnetic_fields(t, 'readout')
    t = open_3D_shutters(t)
    t = camera_delay(t)
    t = readout(t)
```

```
    t = PGC_in_traps(t)
    t = set_magnetic_fields(t, 'MOT')
    t = open_2D_shutter(t)
    t = slow_noise_eater(t)
    #t = magnetic_field_monitor(t)
    #t = set_magnetic_fields(t, 'MOT')
    t = idle(t)
    return t

def Rydberg_pulse(t, time459, time1038, site):
    """Do a Rydberg pulse on one site. This waveform is used inside the Rydberg and Cz waveforms
        .
    It is used do eliminate repetition, but it is not sufficient on its own because it does not
        switch the scanners
    or traps.
    time459: how long the 459 light is on (not including delays)
    time1038 how long the 1038 light is on (not including delays)
    site: a string directing the 459 switch DDS to the correct profile (e.g. 'site 1', 'site 2',
         etc.)"""

    # Rydberg light on
    Rydberg459A(t-Ryd459A_on_delay, site)
    Rydberg1038(t-Ryd1038_on_delay, 'on')
    # Rydberg light off
    Rydberg459A(t+time459+Ryd459A_off_delay, 'off')
    Rydberg1038(t+time1038+Ryd1038_off_delay, 'off')
    # set the time to after 459 and 1038 have both ended
    t += max(rydberg_A_459_time_control, rydberg_A_1038_time_control)
    return t

def Rydberg(t, site):
    """Do a Rydberg pulse on a single site. Choose the site by passing in a string 'site' (e.g.
        'site 1', 'site 2',
    etc.) that will select the appropriate DDS profile."""

    # scanners to correct site
    scanner459_1(t, site)
    scanner459_2(t, site)
    scanner1038_1(t, site)
    scanner1038_2(t, site)

    # timing relative to the start of the pulse
    # calculate which of these delays is the longest, so we wait as little time as possible
        before excitation
    t += max(Ryd1038_on_delay, Ryd1038_scanner1_delay, Ryd1038_scanner2_delay,
            Ryd459A_on_delay, Ryd459_scanner1_delay, Ryd459_scanner2_delay,
            trap780top_off_delay, trap780bottom_off_delay)

    # scope trigger on (it's okay if this happens before the beginning of this waveform)
    HSDIO(t-HSDIO2_delay, 34, 1)
    # 780 traps off
    SHG780(t-trap780top_off_delay, 'off')
    Verdi780(t-trap780bottom_off_delay, 'off')

    ### pi pulse ###
    t = Rydberg_pulse(t, rydberg_A_459_time_control, rydberg_A_1038_time_control, site)
```

```
    # scope trigger off
    HSDIO(t-HSDIO2_delay, 34, 0)

    # 780 traps on
    SHG780(t+trap780top_on_delay, 'on')
    Verdi780(t+trap780bottom_on_delay, 'on')

    # wait until the traps are really back on
    t += delay_after_excitation_pulse

    # scanners off
    t1 = scanner459_1(t, 'off')
    t2 = scanner459_2(t, 'off')
    # wait for DDS delay if necessary
    t = max(t1, t2)

    return t

def Cz(t, control, target):
    """Perform a Cz gate between two sites.
    This is a pi Rydberg pulse on the control site, a 2*pi Rydberg pulse on the target site, and
         a pi Rydberg pulse on
    the control site again.
    'control' and 'target' parameters are strings that direct the DDS profiles to the correct
        sites
    (e.g. 'site 1', 'site 2')."""

    # replaces a 19 x 22 table (418 elements) with

    # scanners to control site
    scanner459_1(t, control)
    scanner459_2(t, control)
    scanner1038_1(t, control)
    scanner1038_2(t, control)

    # timing relative to the start of the pulse
    # calculate which of these delays is the longest, so we wait as little time as possible
        before excitation
    t += max(Ryd1038_on_delay, Ryd1038_scanner1_delay, Ryd1038_scanner2_delay,
            Ryd459A_on_delay, Ryd459_scanner1_delay, Ryd459_scanner2_delay,
            trap780top_off_delay, trap780bottom_off_delay)

    # scope trigger (it's okay if this happens before the beginning of this waveform)
    HSDIO(t-HSDIO2_delay, 34, 1)
    # 780 traps off
    SHG780(t-trap780top_off_delay, 'off')
    Verdi780(t-trap780bottom_off_delay, 'off')

    ### pi pulse on control site ###
    t = Rydberg_pulse(t, rydberg_A_459_time_control, rydberg_A_1038_time_control, control)

    # scope trigger off
    HSDIO(t-HSDIO2_delay, 34, 0)
    # wait until both 459 and 1038 are really off
    t += delay_before_switching

    # scanners to target site
```

```
    scanner459_1(t, target)
    scanner459_2(t, target)
    scanner1038_1(t, target)
    scanner1038_2(t, target)

    # time it takes to fully switch sites
    switching_delay = max(Ryd1038_on_delay, Ryd1038_scanner1_delay, Ryd1038_scanner2_delay,
                          Ryd459A_on_delay, Ryd459_scanner1_delay, Ryd459_scanner2_delay,
                          ) if doGates else 5
    t += switching_delay

    ### 2*pi pulse on target site ###
    t = Rydberg_pulse(t, rydberg_A_459_time_target, rydberg_A_1038_time_target, target)

    # wait until both 459 and 1038 are both really off
    t += delay_before_switching

    # scanners to control site
    scanner459_1(t, control)
    scanner459_2(t, control)
    scanner1038_1(t, control)
    scanner1038_2(t, control)

    t += switching_delay

    ### pi pulse on control site ###
    t = Rydberg_pulse(t, rydberg_A_459_time_control, rydberg_A_1038_time_control, control)

    # 780 traps on
    SHG780(t+trap780top_on_delay, 'on')
    Verdi780(t+trap780bottom_on_delay, 'on')

    # wait until the traps are really back on
    t += delay_after_excitation_pulse

    # scanners off
    t1 = scanner459_1(t, 'off')
    t2 = scanner459_2(t, 'off')
    # wait for DDS delay if necessary
    t = max(t1, t2)

    return t

### different experiments ###

def state_F3(t):
    """Do state prep into F=3. A high signal is expected."""
    t = Rydberg_magnetic_fields(t)
    t = uwave_pi_global(t)
    t = set_magnetic_fields(t, 'readout')
    t = do_blowaway(t)
    return t

def state_F4(t):
    """Do state prep into F=4. A low signal is expected."""
    t = set_magnetic_fields(t, 'readout')
    t = do_blowaway(t)
```

```
    return t

def optimize_traps(t):
    """Use a trap drop, but no blowaway, so we can optimize the trap loading and retention."""
    t = Rydberg_magnetic_fields(t)
    t = uwave_pi_global(t)
    t = set_magnetic_fields(t, 'readout')
    t = do_blowaway(t)
    return t

def state_prep(t):
    """Prepare different state |11>, |10>, |01>, and |00> depending on the value of state_det,
        which should be stepped
    in the independent variables."""

    # Switch to Rydberg fields, unless we're just going to do a no_op.
    if state_det != 0:
        t = Rydberg_magnetic_fields(t)
    # Perform the appropriate microwave pulse
    t = [no_op, uwave_wStark_site2_Pi_Raman, uwave_wStark_site1_Pi_Raman, uwave_pi_global][
        state_det](t)
    # blowaway
    t = set_magnetic_fields(t, 'readout')
    t = do_blowaway(t)
    return t

def CNOT(t):
    # prepare one of four states |11>, |01>, |10>, |00>
    t = [no_op, uwave_wStark_site2_Pi_Raman, uwave_wStark_site1_Pi_Raman, uwave_pi_global][
        input_state](t)
    t = uwave_wStark_site2_Pi2_Raman(t)
    t = Cz(t, 'site 1', 'site 2')
    t = uwave_wStark_site2_Pi2_Raman_phase(t)
    # readout one of four states |00>, |10>, |01>, |11> as a high-high signal
    t = [no_op, uwave_wStark_site2_Pi_Raman, uwave_wStark_site1_Pi_Raman, uwave_pi_global][
        state_det](t)
    t = scanners459_off(t)
    # blowaway
    t = set_magnetic_fields(t, 'readout')
    t = do_blowaway(t)
    return t

def Ground459(t):
    """Perform a 459 Ramsey experiment by doing a 459 pulse sandwiched by two microwave pi/2
        pulses."""
    t = Rydberg_magnetic_fields(t)
    t = uwave_pi_by_2_global(t)
    t = Rydberg(t, 'site 1')
    t = Rydberg(t, 'site 2')
    t = uwave_pi_by_2_global(t)
    t = scanners459_off(t)
    # blowaway
    t = set_magnetic_fields(t, 'readout')
    t = do_blowaway(t)
    return t

def OP_optimize(t):
```

```
    t = OP_depump(t)
    t = set_magnetic_fields(t, 'readout')
    t = do_blowaway(t)
    return(t)

def optimize_0_1(t):
    """Switch between doing a pi pulse and no pi pulse, so you can optimize the high-low signal.
         Includes an OP depumping and trap drop."""
    t = OP_depump(t)
    t = Rydberg_magnetic_fields(t)
    if state==3:
        t = uwave_pi_global(t)
    t = trap_drop(t)
    t = set_magnetic_fields(t, 'readout')
    t = do_blowaway(t)
    return(t)

### The above are all just definitions. Specify the waveforms to actually run below. ###

t = before_experiment(t)
t = optimize_0_1(t)
t = after_experiment(t)
```

D AQuA Cost Functions

```
def CNOT(experimentResults, site1, site2):

    a11 = experimentResults['iterations/0/analysis/loading_retention/atoms']
    a10 = experimentResults['iterations/1/analysis/loading_retention/atoms']
    a01 = experimentResults['iterations/2/analysis/loading_retention/atoms']
    a00 = experimentResults['iterations/3/analysis/loading_retention/atoms']

    # then evaluate if the CNOT works properly when both atoms load
    # by simply adding the number of retained atoms, and not the retained fraction,
    # we can simultaneously optimize loading. The output rotations should be set so
    # that a high-high signal is always desirable.
    # atoms is size 250 x 2 x 49
    cost = (numpy.sum(a11[:,0,site1]*a11[:,0,site2]*a11[:,1,site1]*a11[:,1,site2]) +
               numpy.sum(a10[:,0,site1]*a10[:,0,site2]*a10[:,1,site1]*a10[:,1,site2]) +
               numpy.sum(a01[:,0,site1]*a01[:,0,site2]*a01[:,1,site1]*a01[:,1,site2]) +
               numpy.sum(a00[:,0,site1]*a00[:,0,site2]*a00[:,1,site1]*a00[:,1,site2]) )

    # normalize by number of measurements, and add a minus sign because the optimizer minimizes
    return -cost


def CNOT11(experimentResults, site1, site2):

    a11 = experimentResults['iterations/0/analysis/loading_retention/atoms']

    # then evaluate if the CNOT works properly when both atoms load
    # by simply adding the number of retained atoms, and not the retained fraction,
    # we can simultaneously optimize loading. The output rotations should be set so
    # that a high-high signal is always desirable.
    # atoms is size 250 x 2 x 49
    cost = (numpy.sum(a11[:,0,site1]*a11[:,0,site2]*a11[:,1,site1]*a11[:,1,site2]) )
```

```
    # normalize by number of measurements, and add a minus sign because the optimizer minimizes
    return -cost


def control_loss(experimentResults, site1):
    # maximize retention of the control atom
    return -experimentResults['iterations/0/analysis/loading_retention/retention'].value[site1]


def loading_retention_readout(experimentResults):
    """Our usual loading optimization that also optimizes readout via the histogram overlap.
    Can be used to reduce atom temperature with a trap drop.
    Optical pumping optimization is more sensitive if used with an OP_Depump phase."""

    iterationResults = experimentResults['iterations/0']

    # get number of retained atoms
    retained = iterationResults['analysis/loading_retention/retained'].value

    # get overlap from histogram results
    hist = iterationResults['analysis/histogram_results'].value
    #average overlap over all shots (should produce a 49 element array)
    overlap = numpy.nanmean(hist['overlap'], axis=0)

    # get variable values
    Readout_time = iterationResults['variables/Readout_time'].value
    #total_AO_time = iterationResults['variables/total_AO_time'].value # no longer exists now
        with functional waveforms
    num_measurements = len(iterationResults['measurements'])

    # calculate the cost for each site
    sitecosts=-(retained-num_measurements*overlap)/(Readout_time)

    # average all sites
    return numpy.nanmean(sitecosts)

def minimize_retention(experimentResults):
    """Minimize the retention, such as would be used with an OP_Depump (no microwave) experiment
        ."""

    iterationResults = experimentResults['iterations/0']
    retention = iterationResults['analysis/loading_retention/retention'].value
    return numpy.nanmean(retention)

def high_low(experimentResults):
    """Maximize retention in the 0th iteration, and minimize retention in the 1st iteration.
    Useful for maximizing loading and optical pumping at the same time."""

    loading0 = experimentResults['iterations/0/analysis/loading_retention/loading'].value
    loading1 = experimentResults['iterations/1/analysis/loading_retention/loading'].value
    retention0 = experimentResults['iterations/0/analysis/loading_retention/retention'].value
    retention1 = experimentResults['iterations/1/analysis/loading_retention/retention'].value

    # maximize retention difference, maximize mean loading
    cost = (retention0-retention1)*(loading0+loading1)/2.0
    # average all sites, minus sign to minimize
    return -numpy.nanmean(cost)
```

```
#self.yi = control_loss(experimentResults, 36)
#self.yi = CNOT(experimentResults, 36, 22)
#self.yi = CNOT11(experimentResults, 36, 22)
#self.yi = loading_retention_readout(experimentResults)
#self.yi = minimize_retention(experimentResults)
self.yi = high_low(experimentResults)

# evaluate cost of iteration just finished
#iterationResults = experimentResults['iterations/0']
#num_measurements = len(iterationResults['measurements'])
#iterations = map(int, experimentResults['iterations'].keys())
#iterations.sort()

# 2 layer OP optimizer that takes |1>-|0>
#retention0 = experimentResults['iterations/0/analysis/loading_retention/retention'].value
#retention1 = experimentResults['iterations/1/analysis/loading_retention/retention'].value
#self.yi = -numpy.nanmean(retention1-retention0)

# minimize |0>
#retention0 = experimentResults['iterations/0/analysis/loading_retention/retention'].value
#self.yi = numpy.nanmean(retention0)

## sum up all the loaded atoms from shot 0 in region 24 optimize brightness
## (negative because cost will be minimized, must convert to float otherwise negative wraps
    around)
# self.yi = -numpy.sum(numpy.array([m['analysis/squareROIsums'][0][24] for m in iterationResults
    ['measurements'].itervalues()]), dtype=numpy.float64)

## take the retention in shot 1 for site 31 thresholded
#self.yi = -numpy.sum(numpy.array([m['analysis/squareROIthresholded'][1,31] for m in
    iterationResults['measurements'].itervalues()]))

## take the signal-to-noise in for selected shot and regions
#regions = range(49)
#shot = 0
#roi_size = 3*3 # each roi is 3 by 3 pixels
## create a length 49 array of the roi sums
#roi_sums = numpy.sum(numpy.array([m['analysis/squareROIsums'][shot] for m in iterationResults['
    measurements'].itervalues()]), axis=0)
## sum over either all regions, or just selected ones
#all_region_sum = numpy.sum(roi_sums)
#region_sum = numpy.sum(roi_sums[regions])
## background is the whole shot, except the regions
#all_sum = numpy.sum(numpy.array([m['data/Hamamatsu/shots/'+str(shot)] for m in iterationResults
    ['measurements'].itervalues()]))
#background_sum = all_sum - all_region_sum
## get the size of an image
#all_region_pixels = len(roi_sums)*roi_size # 49 regions
#region_pixels = len(regions)*roi_size # selected regions
#image_shape = numpy.shape(iterationResults['measurements'].values()[0]['data/Hamamatsu/shots/'+
    str(shot)])
#image_pixels = image_shape[0]*image_shape[1]
#background_pixels = image_pixels - all_region_pixels
##normalize by pixels
#signal = region_sum*1.0/region_pixels
```

```
#noise = background_sum*1.0/background_pixels
#self.yi = noise/signal

#take the amplitude of a gaussian fit to the ROIs
#self.yi = -iterationResults['analysis/gaussian_roi/fit_params'].value[4]

#loading & retention
#loaded = iterationResults['analysis/loading_retention/loaded'].value
#retained = experimentResults['iterations/0/analysis/loading_retention/retained'].value
#atoms = experimentResults['iterations/0/analysis/loading_retention/atoms'].value
#loading = numpy.mean(iterationResults['analysis/loading_retention/loading'].value)
#retention = iterationResults['analysis/loading_retention/retention'].value
#self.yi = -numpy.nanmean(retained)

# retention from all iterations
#all_retained = numpy.nanmean(numpy.array([experimentResults['iterations/{}/analysis/
    loading_retention/retained'.format(i)].value for i in iterations]), axis=0)

# histogram
#hist = iterationResults['analysis/histogram_results'].value
#average overlap over all shots (should produce a 49 element array)
#overlap = numpy.nanmean(hist['overlap'], axis=0)
#average all ROIs from shot 0
#overlap = numpy.nanmean(hist[0]['overlap'])

# overlap from all iterations
#all_overlap = numpy.nanmean(numpy.array([experimentResults['iterations/{}/analysis/
    histogram_results'.format(i)].value['overlap'] for i in iterations]), axis=0)

# gaussian fit
# maximize: amplitude
# minimize: wx, wy, goodness of fit
#fit = numpy.sum(numpy.abs(iterationResults['analysis/gaussian_roi/covariance_matrix'].value))
#amplitude = iterationResults['analysis/gaussian_roi/fit_params'].value[4]
#wx = iterationResults['analysis/gaussian_roi/fit_params'].value[5]
#wy = iterationResults['analysis/gaussian_roi/fit_params'].value[6]
#blacklevel = iterationResults['analysis/gaussian_roi/fit_params'].value[8]
#self.yi = fit*wx*wy*blacklevel / amplitude

# use the DC Noise Eater data
# box 0, channel 1, std of error signal
# self.yi = numpy.std([m['data/DC_noise_eater'][0,1,11] for m in iterationResults])

# use the Ramsey curve fit
# ramsey experiment optimizer cost function:
#ramsey_frequency = experimentResults['analysis/Ramsey/frequency'].value
#ramsey_decay = experimentResults['analysis/Ramsey/decay'].value
#self.yi = -ramsey_frequency

#sitecosts=-loading
# take only site 39 with ramsey data
#self.yi = -(all_retained[39]-250*all_overlap[39])/Readout_time - ramsey_frequency -
    ramsey_decay

# return inf instead of nan
if numpy.isnan(self.yi):
    self.yi = numpy.inf
```

E AQuA Report

```
(
'parity phase = {}\n'.format(parity_phase) +

'\n'.join([i+' = '+str(locals()[i]) for i in [
'scanner_1_frequency_1_459',
'scanner_2_frequency_1_459',
'scanner_1_frequency_1_1038',
'scanner_2_frequency_1_1038',
'scanner_1_frequency_2_459',
'scanner_2_frequency_2_459',
'scanner_1_frequency_2_1038',
'scanner_2_frequency_2_1038'
]]) +

'\n\n' +

'\n'.join([i+' = '+str(locals()[i]) for i in [
'RydA_switch_AOM_frequency_1',
'RydA_switch_AOM_frequency_2',
'site_1_RydA_frequency_offset',
'site_2_RydA_frequency_offset',
'sitetosite_difference',
'rydberg_A_459_time_control',
'rydberg_A_459_time_target',
'microwave_pi_pulse',
'blow_away_time',
'T2_delay'
]])+

,,,

** Rydberg **
level: {}{}_{},{}
polarization:
\t459:
\t1038:
bias magnetic field during gate: 1.5G
beam waists:
\t459: {} (um)
\t1038 {} (um)
power at atoms:
\t459A: {}
\t1038: {}
frequencies:
\tdetuning from center of mass: {} (GHz)
\tRabi:
\t\t459a: (MHz)
\t\t459b: (MHz)
\t\t1038: (MHz)
\t\tRydberg: {} (MHz)
spontaneous emission errors:
\tRydberg 2pi time: (us)
\t7p lifetime: (us)
\tRydberg lifetime: {} (us)
\tProbability of spontaneous decay from 7p in pi pulse:
```

```
\tProbability of Rydberg spontaneous decay in 2pi time:
'''.format(
Rydberg_n,
['S','P','D','F'][Rydberg_l],
asHalf(Rydberg_j),
asHalf(Rydberg_m),
Rydberg_459_beam_waist,
Rydberg_1038_beam_waist,
Rydberg_459_A_power_at_atoms,
Rydberg_1038_power_at_atoms,
Rydberg_detuning,
Rydberg_Rabi_Frequency_over_2pi * 2*pi,
1/Rydberg_sponteneous_emission_rate
)

+'\nblow_away_time = {}\n'.format(blow_away_time)

)
```