APC 524 Final Project

Alira Laser Software

Paris Blaisdell-Pijuan
Jacob Bryon
Youssef Elasser
Zachary Hervieux-Moore
Joseph Sartini

Summary:

The Alira Laser Software project is based on a former project in Paris' lab called Alira Biosensing whose purpose was to develop a non-invasive glucose sensor. As you may know, glucose sensing is a major part of living with diabetes and even current sensors requires a prick of the finger to draw blood. Paris' lab is working on sensing techniques using lasers to alleviate the need of drawing blood and making the entire process non-invasive. To develop this sensor, Paris is currently working with a laser setup that is quite cumbersome due to the variety of third party data recorders and laser drivers. The current third party software only supports vary basic scans with the laser and doing the desired laser tests are difficult to perform. However, there exists a combination of Python packages and .dll's to control all the instruments. The goal of this project is to bundle these drivers into a custom piece of software to make defining of experiments and performing on the laser a seamless process without a need to have any knowledge of the laser driver software.

Desired Functionality:

The software in question has the following three components:

- A simple programmable interface to the laser that handles the setup, operation, and tear down of the laser.
- An experiment API that makes the defining of experiments simple and exogenous of the laser operation.
- A data analysis package and GUI that aids the post data-collection analysis of the laser sensor data

In particular

Laser Interface:

- Should handle connecting to the sensor
- Validate user inputs to the laser
- Abstract the .dll's functions into useful high level functions
- Handle the collecting and saving of the laser data
- Handle the turning off of the sensor in a safe and robust manner

Experiment API:

- Should be completely separate of the laser interface - the user should not have to turn the laser on and off
- Should have a simple and compact API but have the expressiveness of being able to complete any series of laser parameter manipulation

Data Analysis Package:

- Contain a GUI application to explore the data visually
- Provide basic numerical methods to garner metrics on the data

External Packages:

The project makes use of the following external libraries:

- zhinst - A Python package developed by Zurich Instruments to interface with their measuring equipment
- SidekickSDK - A set of dll's developed by SideKick for their laser controller
- Matplotlib - Primarily use in the creation of the GUI
- Finally, a custom piece of software developed in Paris' lab that served as the skeleton to the laser interface

Classes:

The UML for the classes is shown on the next page. The functionality is described here.

Laser Interface:

The laser interface controls the laser. It simply exposes functions to the user that they can set parameters to values within validated ranges. For example, if the user wishes for a wavelength of 100nm on the laser, they would call laser.set_field('wavelength', 100). However, this requires a series of instructions to the laser controller that the user should remain oblivious to.

The laser class is desired to be a singleton class. This is because there is only one physical laser in the real world. Thus, a singleton class is desired to prevent any accidental duplication of the interface and having competing signals sent to the controller which may cause damage to the controller.

Experiment API:

The experiment API wishes to have an API that is expressive and compact. The form settled on was

```
Experiment.builder().with_actions([actions]).with_duration(10).build()
```

The API reading as follows: retrieve the builder for the Experiment class, assign actions to be performed every second, assign a duration to the experiment, finally build the desired Experiment class.

The reasoning is as follows. Experiments (the physical world meaning) are simply created by a sequence of steps. There is only one abstract notion of experiment in the English language, the only thing that differs between experiments are the steps (what we call actions) performed. Thus, we have designated the Experiment class to follow the builder design pattern. This makes validation of plausible experiments easy in the .build() step. Thus, the Experiment class offloads the heavy lifting to the Action class.

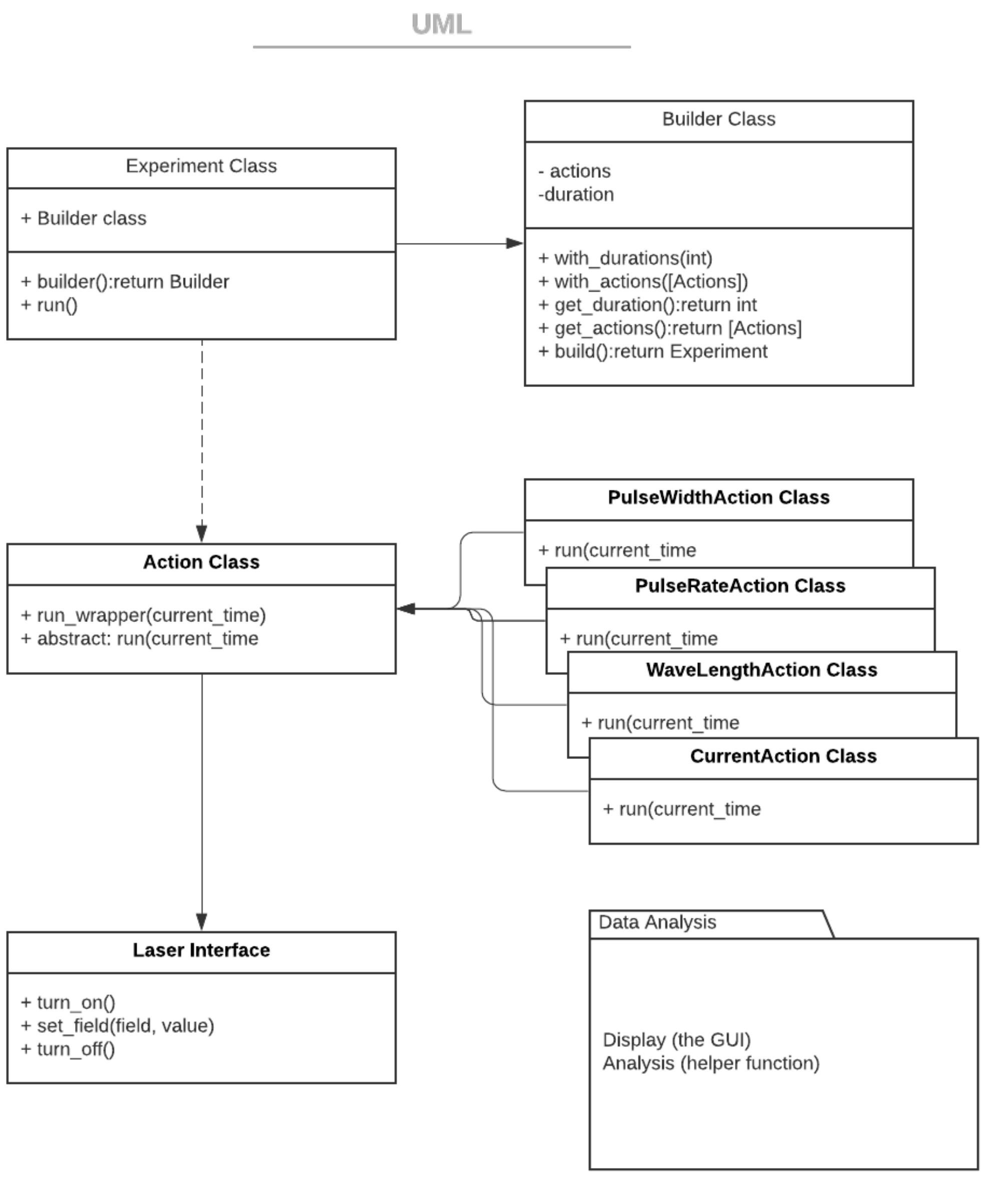
Action Class:

This is an abstract base class that implements requires the user to implement run(current_time) which simply must return the value that the desired action should perform on the laser. Thus, the user is expected to subclass one of PulseRateAction, PulseWidthAction, WavelengthAction, or CurrentAction and implement the run(current_time) whose return value sets one of the four parameters of the laser at the desired wall clock time of current_time (seconds since start of experiment). The user specified run(…) function is wrapped in run_wrapper(current_time) which handles retrieving the value and interfacing with the laser. That is the Experiment class simply feeds in the current_time to its list of actions and runs run_wrapper(…) on each of its actions. This achieves decoupling of the Experiment class and the Laser interface.

Data Analysis Package

As alluded to before, this package is a simple GUI interface that allows for quick visualization of the output. It also provides a bunch of helper functions to perform numerical operations on the data to derive metrics in the GUI.

UML Diagram:


UML
Builder Class
- actions
-duration
+ with_durations(int)
+ with_actions([Actions])
+ get_duration():return int
+ get_actions():return [Actions]
+ build():return Experiment
Experiment Class
+ Builder class
+ builder():return Builder
+ run()
PulseWidthAction Class
+ run(current_time
PulseRateAction Class
+ run(current_time
WaveLengthAction Class
+ run(current_time
CurrentAction Class
+ run(current_time
Action Class
+ run_wrapper(current_time)
+ abstract: run(current_time
Laser Interface
+ turn_on()
+ set_field(field, value)
+ turn_off()
Data Analysis
Display (the GUI)
Analysis (helper function)