Contents

1 Project Organization

Figure 1 shows how the project is orgaized.

- The `data` directory contains the raw data collected.
  - `delay` contains data on the behavior and performance of the delay modules
  - `digipot` contains information on the AD5206 digital potentiometer (a chip you probably want to avoid)
  - `filters` has information on the bandpass filters and their characteristics over various frequency ranges
  - `lasers` has current-power curves for the three lasers in Lucas' lab.
  - `machzehnder` has information on the response of the three MZMs.
- `doc` has this document, and associated source files.
- `figures` has various figures and results, derived from the data. It has the same categories as the `data` directory.
  - The `pinouts` directory has pinouts for the various ICs I used.
- `manuals` has PDF manuals for the various bits used in the circuit.
- `notes` has application notes and other writeups. These also have copies in the lab notebook.
- `numericalanalysis` has my work on numerical bifurcation and theoretical prediction of the size and shape of the amplitude death region.
- `papers` has PDF copies of various relevant papers.
- `scripts` has all the Python source code for collecting and analyzing data. I'll got into more depth on what each of these scripts do later.

2 Data Format

All data is stored in the Numpy builtin compressed-archive format (`.npz`), since it's easy to work with in Python, makes it possible to store the data and metadata about its capture (such as time, test parameters, etc.) in the same file, and is nice and compact (no more 80MB captures!).

2.1 Saving and Loading

I've written a few utility functions to make saving and loading these files as simple as possible. Located in `utils.py`, the `save` function takes a filename, arbitrary Numpy array (any shape or dimension number), and (optionally) a dictionary of settings to save; and writes them to a file.

Similarly, the `load` function, given a filename, loads that file and returns a `data, metadata` tuple. If the metadata is stored in a separate file (which I was doing for the first few weeks), this function loads from that instead, and resaves the data in compressed form. The code snippet below shows how to use these functions:

```
import utils

data = get_data()
metadata = {
    'capname': 'Example Capture,
```

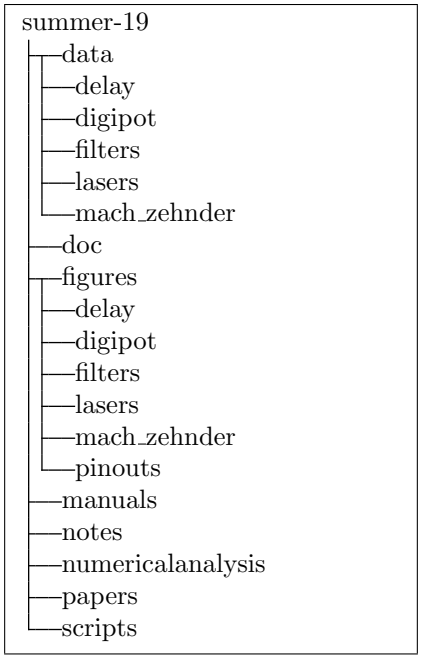

Figure 1: The directory format of the project.

```
    'lasernum': 3,
    'lasername': 'Clara
}

utils.save("outfile", data, metadata)


# Then later, when you need the data back:

data, md = utils.load("outfile")
```

2.2 Editing Files

The script `edit_md.py` can be used to modify a file's settings. Run it as

```
python scripts/edit_md.py filename
```

and follow the instructions on the screen.

3 Settings

Various settings for the project can be found in `settings.py`. The first three entries in the file (`ROOT_DIR`, `FIGS_DIR`, and `DATA_DIR`) set the working directory, and directories to save and load figures and data from, respectively. The snippet of Python code for `ROOT_DIR`

```
ROOT_DIR = Path(__file__).parents[1]
```

simply sets the root directory to the parent of the scripts directory. You shouldn't need to change any of these parameters, short of a major reorganization of the project.

The rest of the settings are snippets from the VISA ids of various instruments used:

- The scope is a Tektronix TDS-2000C 4-channel scope w/ USB out. There's one (and only one) of these in the JLab.
- The function generator is an Agilent 33210A.
- The multimeter is an

I believe these snippets will be the same for any instrument of the same type, but I am not sure. If you get "InstrumentNotFoundError" or similar, this may be the cause.

4 Characterizing MZMs

To characterize an MZM, connect it to the associated laser, and turn on the laser power. Feed a $10V_{pp}$ triangle wave into the MZM at 8Hz. Connect channel 2 on the scope to the function generator, and channel 1 to the detector output. Run

```
python scripts/MZ_getdata.py
```

and follow the prompts for filename and power level.

After collecting this capture, run

```
python scripts/MZ_characterize.py <filename>
```

to fit the appropriate curve. The fitted parameters will be printed to the terminal, and the plot saved to the figures directory. Make sure to inspect the plot to verify that the fit actually worked!

The number you probably most care about is the $V\pi$ figure reported on the terminal—this can be compared to the value on the datasheet, and used to determine the necessary offset voltages.

5 Characterizing Lasers

Unfortunately, this is the least-automated procedure in the whole project. Because I wasn't able to find out how to programmatically set the laser controller (I believe it's possible, but would require an Arduino or something), it requires manually measuring voltage at a variety of input currents. That being said, the procedure is:

1. Connect the output of the laser to the detector, without the polarization controller or MZM in the fiber path.
2. Connect the output of the detector to a multimeter set to measure DC voltage.
3. Open a spreadsheet, and create two columns (current and voltage).
4. At various input currents, measure the output voltage
   (a) There is a threshold current, typically about 8mA
   (b) My measurements were spaced at 1mA, but decent results should be achievable with slightly wider spacing
   (c) There is an overcurrent protection around 50mA, but you shouldn't need to go that high anyways—typically, current while running experiments is ca. 20mA.
5. Save the spreadsheet (as a CSV) in `data/lasers`
6. Create a `.dat` file for the laser and detector configuration. Entries in this file:
   (a) `name`: The nickname given to the laser (Alice, Clara, and Kira)

(b) `wavelength`: The peak wavelength, as given by the spec sheet

(c) `R`: The detector efficiency at that wavelength, as given by the detector spec sheet

(d) `G`: The gain setting of the detector

(e) `Threshold`: The expected threshold current, from the datasheet.

(f) `OpCurrent`: The expected current-power relation, in mW/40mA. Also from the datasheet.

Use one of the existing files as a formatting guide.

6 Characterizing Delay Module

For the delay module, we want to know whether and how the input signal is distorted and shifted (apart from the time delay, obviously). Some tests, such as determining the maximum input voltage and observing distortion of non-sinosoidal signals, are purely qualitative, and need little explanation.

Quantitative tests we can perform on the delay module include measuring its gain and phase shift across a variety of frequencies.