
ac-training-lab Documentation

Release 0.0.post1.dev60+g7cbe265

Sterling G. Baird

May 15, 2024

CONTENTS

1	Workflows	5
1.1	Dedicated Workflows	5
1.2	Supported Workflows	6
2	Functionality	7
3	Feedback	9
4	Contents	11
4.1	Contributing	11
4.2	License	16
4.3	Contributors	17
4.4	ac_training_lab	17
5	Indices and tables	19
	Python Module Index	21
	Index	23

Warning: This is an ongoing project. If you would like to participate or are interested in contributing, please [introduce yourself](#) or reach out to sterling.baired@utoronto.ca.

The [Acceleration Consortium](#) (AC) Training Lab is a remotely accessible facility that houses a diverse set of physical hardware for self-driving laboratories (SDLs) including liquid handlers, solid dispensers, Cartesian-axis systems, mobile robotic arms, and synthesis and characterization modules. Where possible, both educational and research-grade hardware are included. The AC Training Lab is used to develop and test SDLs and to provide a platform for training students and researchers in the use of SDLs. The [AC Training Lab GitHub repository](#) also acts as an example of setting up an autonomous laboratory.

Star Follow @AccelerationConsortium Follow @sgbaird Issue Discuss

The equipment in the training lab can be broadly categorized in the following categories: characterization, prototyping, synthesis, dispensing, environment, and infrastructure. See the image below for an example of some of the equipment intended for the AC Training Lab.

Characterization



Dispensing



Prototyping



Environment
















Synthesis



Infrastructure



Here are some of the modules we have procured and are in the process of setting up (help wanted if you're in Toronto!):

Name	Image	Qty	Description	Video
Science Jubilee		1(+4)	A versatile, open-source toolchanger with a large community of users and developers which is used for both general 3D printing and scientific applications. One nearing completion. Four more planned.	
Delta Stage Microscope		2	A DIY open-source microscope with a fine-positioning, motorized stage. Reflection illumination version of Delta Stage	
Opentrons OT-2		1	An open-source and cost-friendly commercial liquid handler	
Opentrons Flex		1	An open-source commercial liquid handler tailored towards high-throughput and advanced liquid handling operations	
Hiwonder ArmPi FPV		1	An educational six-axis robotic arm	
Hiwonder JetAuto Pro		1	An educational six-axis mobile cobot with a 3D depth camera and lidar	
MyCobot 280 and MyAGV		1	An educational six-axis mobile cobot	
AutoTrickler v4		1	An automated solid dispensing station, usually marketed for ammunition reloading, but to be used as a general-purpose powder doser	
Charge-Master Supreme		1	An automated solid dispensing station, usually marketed for ammunition reloading, but to be used as a general-purpose powder doser	
Ingenuity Powder System		1	An automated solid dispensing station, usually marketed for ammunition reloading, but to be used as a general-purpose powder doser	
MT Powder Powder Doser		1	XPR105DU is a commercial, automated powder doser by Mettler-Toledo	
Cocoa		1	A commercially sold and mostly open-source chocolate 3D printer kit	
2Press				CONTENTS
FormAuto and Form 3+		1	24/7 autonomous SLA 3D printer with camera inspection	




Name	Im- age	Qty	Description
Mobile manipulator		1	A research-grade six-axis mobile cobot with vision capabilities optimized for laboratory environments
Desktop SEM		1	A desktop scanning electron microscope (SEM) with Python integrations
Chamber interfaces (TBD)		-	e.g., miniature glovebox, miniature ductless fumehood, small nitrogen generator
Low-force tensile tester		1	Low-cost, open-source tensile tester. Examples [1], [2], [3], [4]

WORKFLOWS

The AC Training Lab is intended as a hands-on sandbox and prototyping environment for researchers. Each workflow will either be dedicated (permanent) or supported (non-permanent).

1.1 Dedicated Workflows

While the equipment is not restricted to particular workflows, we are actively developing a subset of readily accessible workflows for the AC Training Lab. Note that single workflow could be carried out using different sets of equipment within the training lab. These workflows will use dedicated hardware in a permanent setup to allow for 24/7 access. The core workflows that are planned, in development, or available are listed below:

Name	Diagram	Description	Status
Light-based color matching		Adjust red, green, and blue LED power levels to match a target color	Ready
Liquid-based color matching		Adjust diluted red, yellow, and blue food coloring pumping power to match a target color	Ready
Solid-based color matching		Adjust the composition of red, yellow, and blue powder (e.g., wax) and processing conditions to match a target color	Development
Chocolate tensile testing		Adjust the composition and processing conditions of 3D printed chocolate tensile specimens to tune the microstructure for maximization of tensile strength	Development
Yeast growth		Adjust reactor temperature to maximize yeast growth and explore nonlinear effects	Development
Titration		Add a base of known concentration to an acid to find the equivalence point as determined by successive pH measurements	Development
Conductivity		Adjust the ratio of battery electrolyte reagents to maximize conductivity and redox potential for a target pH	Planning
Polymer cross-linkage			Planning

1.2 Supported Workflows

Supported workflows (i.e., non-permanent setups) that are planned, in development, or available are listed below:

Name	Dia-gram	Description	Status
Alkaline Catalysis Lifecycle Testing		Adjust the stress-cycling conditions of a nickel electrode in a KOH solution to investigate the cause of catalyst degradation	De-velop-ment
Material recycling		Incorporate the use of “waste” experimental samples as part of a recycling workflow using mixed red, yellow, and blue solid powders	De-velop-ment

FUNCTIONALITY

This refers to the infrastructure-focused capabilities showcased in the AC Training Lab. The core functionalities (intended as permanent demos) that are planned, development, or available are listed below. These functionalities may either be standalone or part of the workflows listed above.

Name	Diagram	Description	Status
Vial transfer (stationary)		Move a vial between adjacent modules	Ready
Vial transfer (mobile)		Move a sample to a different location	Development
Vial capping/decapping		Cap or decap a vial	Development
Tool changing		Swap a tool on a robotic arm	Development

CHAPTER THREE

FEEDBACK

We would love to get suggestions on the [types of workflows and functions](#) you'd like to see in the AC Training Lab! For additional training opportunities offered by the Acceleration Consortium, please navigate to [AC Microcourses](#).

CONTENTS

4.1 Contributing

Welcome to `ac-training-lab` contributor's guide.

This document focuses on getting any potential contributor familiarized with the development processes, but [other kinds of contributions](#) are also appreciated.

If you are new to using [git](#) or have never collaborated in a project previously, please have a look at [contribution-guide.org](#). Other resources are also listed in the excellent [guide created by FreeCodeCamp](#)¹.

Please notice, all users and contributors are expected to be **open, considerate, reasonable, and respectful**. When in doubt, [Python Software Foundation's Code of Conduct](#) is a good reference in terms of behavior guidelines.

4.1.1 Issue Reports

If you experience bugs or general issues with `ac-training-lab`, please have a look on the [issue tracker](#). If you don't see anything useful there, please feel free to fire an issue report.

Tip: Please don't forget to include the closed issues in your search. Sometimes a solution was already reported, and the problem is considered **solved**.

New issue reports should include information about your programming environment (e.g., operating system, Python version) and steps to reproduce the problem. Please try also to simplify the reproduction steps to a very minimal example that still illustrates the problem you are facing. By removing other factors, you help us to identify the root cause of the issue.

4.1.2 Documentation Improvements

You can help improve `ac-training-lab` docs by making them more readable and coherent, or by adding missing information and correcting mistakes.

`ac-training-lab` documentation uses [Sphinx](#) as its main documentation compiler. This means that the docs are kept in the same repository as the project code, and that any documentation update is done in the same way as a code contribution. The documentation uses [CommonMark](#) with [MyST](#) extensions.

¹ Even though, these resources focus on open source projects and communities, the general ideas behind collaborating with other developers to collectively create software are general and can be applied to all sorts of environments, including private companies and proprietary code bases.

Tip: Please notice that the [GitHub web interface](#) provides a quick way of propose changes in ac-training-lab's files. While this mechanism can be tricky for normal code contributions, it works perfectly fine for contributing to the docs, and can be quite handy.

If you are interested in trying this method out, please navigate to the docs folder in the source [repository](#), find which file you would like to propose changes and click in the little pencil icon at the top, to open [GitHub's code editor](#). Once you finish editing the file, please write a message in the form at the bottom of the page describing which changes have you made and what are the motivations behind them and submit your proposal.

When working on documentation changes in your local machine, you can compile them using `tox` :

```
tox -e docs
```

and use Python's built-in web server for a preview in your web browser (<http://localhost:8000>):

```
python3 -m http.server --directory 'docs/_build/html'
```

4.1.3 Code Contributions

A list of projects is available on the documentation [homepage](#).

Project Organization

— AUTHORS.md	<- List of developers and maintainers.
— CHANGELOG.md	<- Changelog to keep track of new features and fixes.
— CONTRIBUTING.md	<- Guidelines for contributing to this project.
— Dockerfile	<- Build a docker container with `docker build .`.
— LICENSE.txt	<- License as chosen on the command-line.
— README.md	<- The top-level README for developers.
— configs	<- Directory for configurations of model & application.
— data	
— external	<- Data from third party sources.
— interim	<- Intermediate data that has been transformed.
— processed	<- The final, canonical data sets for modeling.
— raw	<- The original, immutable data dump.
— docs	<- Directory for Sphinx documentation in rst or md.
— environment.yml	<- The conda environment file for reproducibility.
— models	<- Trained and serialized models, model predictions, or model summaries.
— notebooks	<- Jupyter notebooks. Naming convention is a number (for ordering), the creator's initials and a description, e.g. `1.0-fw-initial-data-exploration`.
— pyproject.toml	<- Build configuration. Don't change! Use `pip install -e .` to install for development or to build `tox -e build`.
— references	<- Data dictionaries, manuals, and all other materials.
— reports	<- Generated analysis as HTML, PDF, LaTeX, etc.
— figures	<- Generated plots and figures for reports.
— scripts	<- Analysis and production scripts which import the actual PYTHON_PKG, e.g. train_model.
— setup.cfg	<- Declarative configuration of your project.

(continues on next page)

(continued from previous page)

```

├── setup.py                <- [DEPRECATED] Use `python setup.py develop` to install for
                             development or `python setup.py bdist_wheel` to build.
├── src
│   └── ac_training_lab    <- Actual Python package where the main functionality goes.
├── tests                  <- Unit tests which can be run with `pytest`.
├── .coveragerc            <- Configuration for coverage reports of unit tests.
├── .isort.cfg             <- Configuration for git hook that sorts imports.
└── .pre-commit-config.yaml <- Configuration of pre-commit git hooks.

```

Submit an issue

Before you work on any non-trivial code contribution it's best to first create either a feature request in the [issue tracker](#) or a new discussion in the [discussions](#) page on the subject. This often provides additional considerations and avoids unnecessary work.

Create an environment

Before you start coding, we recommend creating an isolated [virtual environment](#) to avoid any problems with your installed Python packages. This can easily be done via [Miniconda](#):

```
conda env create -f environment.yml
conda activate ac-training-lab
```

NOTE: The conda environment will have ac-training-lab installed in editable mode. Some changes, e.g. in `setup.cfg`, might require you to run `pip install -e .` again.

Optional and needed only once after `git clone`:

1. install several [pre-commit](#) git hooks with:

```
pre-commit install
# You might also want to run `pre-commit autoupdate`
```

and checkout the configuration under `.pre-commit-config.yaml`. The `-n`, `--no-verify` flag of `git commit` can be used to deactivate pre-commit hooks temporarily.

Clone the repository

The instructions below assume that you are using git's command line interface. Alternatively, you may use [GitHub Desktop](#) or the built-in git functionality of your favorite IDE, such as [VS Code's Source Control extension](#).

1. Create an user account on GitHub if you do not already have one.
2. Fork the project [repository](#): click on the *Fork* button near the top of the page. This creates a copy of the code under your account on GitHub.
3. Clone this copy to your local disk:

```
git clone git@github.com:YourLogin/ac-training-lab.git
cd ac-training-lab
```

4. You should run:

```
pip install -U pip setuptools -e .
```

to be able to import the package under development in the Python REPL.

5. Install `pre-commit`:

```
pip install pre-commit
pre-commit install
```

`ac-training-lab` comes with a lot of hooks configured to automatically help the developer to check the code being written.

Implement your changes

1. Create a branch to hold your changes:

```
git checkout -b my-feature
```

and start making changes. Never work on the main branch!

2. Start your work on this branch. Don't forget to add `docstrings` to new functions, modules and classes, especially if they are part of public APIs.
3. Add yourself to the list of contributors in `AUTHORS.md`.
4. When you're done editing, do:

```
git add <MODIFIED FILES>
git commit
```

to record your changes in `git`.

Please make sure to see the validation messages from `pre-commit` and fix any eventual issues. This should automatically use `flake8/black` to check/fix the code style in a way that is compatible with the project.

Important: Don't forget to add unit tests and documentation in case your contribution adds an additional feature and is not just a bugfix.

Moreover, writing a `descriptive commit message` is highly recommended. In case of doubt, you can check the commit history with:

```
git log --graph --decorate --pretty=oneline --abbrev-commit --all
```

to look for recurring communication patterns.

5. Please check that your changes don't break any unit tests with:

```
tox
```

(after having installed `tox` with `pip install tox` or `pipx`).

You can also use `tox` to run several other pre-configured tasks in the repository. Try `tox -av` to see a list of the available checks.

Submit your contribution

1. If everything works fine, push your local branch to the remote server with:

```
git push -u origin my-feature
```

2. Go to the web page of your fork and click “Create pull request” to send your changes for review.

Find more detailed information in [creating a PR](#). You might also want to open the PR as a draft first and mark it as ready for review after the feedbacks from the continuous integration (CI) system or any required fixes.

Troubleshooting

The following tips can be used when facing problems to build or test the package:

1. Make sure to fetch all the tags from the upstream [repository](#). The command `git describe --abbrev=0 --tags` should return the version you are expecting. If you are trying to run CI scripts in a fork repository, make sure to push all the tags. You can also try to remove all the egg files or the complete egg folder, i.e., `.eggs`, as well as the `*.egg-info` folders in the `src` folder or potentially in the root of your project.
2. Sometimes `tox` misses out when new dependencies are added, especially to `setup.cfg` and `docs/requirements.txt`. If you find any problems with missing dependencies when running a command with `tox`, try to recreate the `tox` environment using the `-r` flag. For example, instead of:

```
tox -e docs
```

Try running:

```
tox -r -e docs
```

3. Make sure to have a reliable `tox` installation that uses the correct Python version (e.g., 3.7+). When in doubt you can run:

```
tox --version
# OR
which tox
```

If you have trouble and are seeing weird errors upon running `tox`, you can also try to create a dedicated [virtual environment](#) with a `tox` binary freshly installed. For example:

```
virtualenv .venv
source .venv/bin/activate
.venv/bin/pip install tox
.venv/bin/tox -e all
```

4. `Pytest` can [drop you](#) in an interactive session in the case an error occurs. In order to do that you need to pass a `--pdb` option (for example by running `tox -- -k <NAME OF THE FALLING TEST> --pdb`). You can also setup breakpoints manually instead of using the `--pdb` option.

4.1.4 Maintainer tasks

Releases

If you are part of the group of maintainers and have correct user permissions on [PyPI](#), the following steps can be used to release a new version for `ac-training-lab`:

1. Make sure all unit tests are successful.
 2. Tag the current commit on the main branch with a release tag, e.g., `v1.2.3`.
 3. Push the new tag to the upstream repository, e.g., `git push upstream v1.2.3`
 4. Clean up the `dist` and `build` folders with `tox -e clean` (or `rm -rf dist build`) to avoid confusion with old builds and Sphinx docs.
 5. Run `tox -e build` and check that the files in `dist` have the correct version (no `.dirty` or `git` hash) according to the `git` tag. Also check the sizes of the distributions, if they are too big (e.g., > 500KB), unwanted clutter may have been accidentally included.
 6. Run `tox -e publish -- --repository pypi` and check that everything was uploaded to [PyPI](#) correctly.
-

4.2 License

The MIT License (MIT)

Copyright (c) 2024 Sterling G. Baird

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

4.3 Contributors

- Sterling G. Baird sterling.baird@utoronto.ca

4.4 ac_training_lab

4.4.1 ac_training_lab package

Module contents

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

a

`ac_training_lab`, [17](#)

INDEX

A

ac_training_lab
 module, [17](#)

M

module
 ac_training_lab, [17](#)