plico motor

Release 0.0.4

Arcetri Adaptive Optics

December 29, 2021

CONTENTS

Release 0.0.4

Date December 29, 2021

plico_motor is an application to control motors.

CHAPTER

ONE

DEVELOPER DOCUMENTATION

If you're interested in contributing to plico_motor, start here:

1.1 Development

1.1.1 Python version

- Python 3.6 or later

Builds on github automatically run all tests on Python versions 3.6, 3.7 and 3.8.

1.1.2 (human) Language

All functions, classes, and especially documentation shall be in English. Use a spell checker.

1.1.3 Coding style

- Style should follow PEP8: https://www.python.org/dev/peps/pep-0008/
- Documentation is generated automatically from docstrings, that should follow the numpydoc convention: https://numpydoc.readthedocs.io/

More details at https://github.com/ArcetriAdaptiveOptics/arte/blob/master/arte/code_convention.py

1.1.4 Developing guide

- Register on github.com and ask to be added to the list of arte's contributors
- Install git on your computer, and configure it with your name and contact email.
- Make sure that you have Python version 3.6+ installed.

Once this has been sorted out, here are the main steps:

Clone the repository

Clone the github repository on your computer:

- `git clone https://github.com/ArcetriAdaptiveOptics/plico_motor.git`
- `cd plico_motor`

Create a feature branch

Unless your change is trivial, it's best to create a branch to work independently:

```
git checkout -b mybranch
```

Develop

Make your changes and commit them into your branch. You can make as many commits as you want in your branch. Make sure that each commit has a readable description, because once developing is complete it will appear in the commit history for all to see, together with your name.

```
git commit -m "descriptive message"
```

Note: These commits are only visible on your computer and not to others, until you merge as described later. Moreover, since they are only kept on your local filesystem, there is no backup unless you provide one yourself.

Make a test

Each feature should have a unit test that verifies the functionality. Test files are in arte/test and follow the same structure as the main tree.

Execute tests with pytest and make sure that none are failing:

```
pytest
```

Keep your branch updated

Due to the distributed nature of git, it is possible that updates are made to the library while you are developing. It is recommended that from time to time you update your branch with these changes.

- `git checkout master` - switch to master branch
- `git pull` - pull all new changes from the github repository
- `git checkout mybranch` - switch back to your branch
- `git merge master` - merge changes from trunk into your branch

If someone has modified the same files as you, a conflict will arise at this point. You have to edit the files to a satisfactory resolution, and commit the result.

Publish

Once the feature is complete, it's time to make it available to others. Assuming that the previous steps are complete:

- all your changes are committed into your branch
- updates from master has been merged
- all tests pass

use this sequence to publish:

- `git checkout master` - switch to master branch
- `git merge mybranch` - merge changes from your branch into master
- `git push` - update master branch on github

If the branch have been kept updated, no conflicts should arise during merge.

After each push, github automatically runs all tests on a virtual machine. To see the result, click on the small icon next to the "Latest commit…" line on the right. Documentation at https://arte.readthedocs.io/en/latest/index.htm is also automatically updated.

Delete branch (optional)

Once the new feature has been merged to master, there is no need to keep the branch around:

- `git branch -d mybranch`

If you plan to further develop the feature, you can keep the branch and go on committing and merging as before.

CHAPTER

TWO

API REFERENCE

The exact API of all functions and classes, as given by the docstrings. The API documents expected types and allowed features for all functions, and all parameters available for the algorithms.

2.1 client

2.1.1 Module contents

2.1.2 Submodules

2.1.3 plico_motor.client.motor_client module

class plico_motor.client.motor_client.**MotorClient**(*rpcHandler*, *sockets*, *axis=1*)

Bases: *plico_motor.client.abstract_motor_client.AbstractMotorClient*, plico.client.hackerable_client.HackerableClient, plico.client.serverinfo_client.ServerInfoClient

home(*timeout_in_sec=10.0*)

move_by(*position_in_steps*, *timeout_in_sec=10.0*)

move_to(*position_in_steps*, *timeout_in_sec=10.0*)

position()

snapshot(*prefix*, *timeout_in_sec=3.0*)

status(*timeout_in_sec=3.0*)

2.1.4 plico_motor.client.abstract_motor_client module

class plico_motor.client.abstract_motor_client.**AbstractMotorClient**

Bases: object

abstract home()

abstract move_by(*position_in_steps*)

abstract move_to(*position_in_steps*)

abstract position()

abstract snapshot(*prefix='MY_MOTOR'*)

abstract status()

2.2 types

2.2.1 Submodules

2.2.2 plico_motor.types.motor_status

class plico_motor.types.motor_status.**MotorStatus**(*name*, *position*, *steps_per_SI_unit*, *was_homed*, *motor_type*, *is_moving*, *last_commanded_position*, *axisno*)

Bases: object

TYPE_LINEAR = 'linear'

TYPE_ROTARY = 'rotary'

as_dict()

2.2.3 Module contents

2.3 utils

2.3.1 Submodules

2.3.2 plico_motor.utils.timeout module

class plico_motor.utils.timeout.**Timeout**

Bases: object

GETTER = 3.0

SETTER = 10.0

2.3.3 Module contents

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX