pylocating

Release 0.2.2

October 26, 2015

Contents

Contents:

CHAPTER 1

Overview

1.1 pylocating

docs	
tests	
package	

Detect the accurate object position disturbed by noise using Artificial Intelligence algorithms.

- Free software: GPLv2 license

1.1.1 Installation

```
pip install pylocating
```

1.1.2 Documentation

https://pylocating.readthedocs.org/

1.1.3 Development

To run the all tests run:

```
tox
```

CHAPTER 2

Installation

At the command line:

```
pip install pylocating
```

CHAPTER 3

Usage

To use pylocating in a project:

```
import pylocating
```

CHAPTER 4

Examples

Note: You should always start examples from root directory of pylocating.

4.1 Federated Particles

Two separated environments contain a different number of particles. All the particles are *PSOParticle* (they follow a standard PSO model). The initial position of particles are around the beacons.

```
/path/to/pylocating$ scripts/bestfitnessgraph.sh federated_particles 20 10
```

arguments:

- 20: the first environment contains 20 particles
- 10: the second environment contains 10 particles

4.2 FollowBest Particles

Two connected environments contain a different number of particles. The first environment contains *GlobalBestPSOParticle* particles (the same of *PSOParticle*, but in this case the best fitness is the best found by all environments instead of the best found inside the environment itself). The second environment contains *FollowBestParticle*; they are special particles that only search around the globally found best position in that moment.

```
/path/to/pylocating$ scripts/bestfitnessgraph.sh followbest_particles 20 10
```

arguments:

- 20: the first environment contains 20 particles
- 10: the second environment contains 10 particles

4.3 Start from beacon sphere surface

One single environment contains all particles. Them are equally distributed around the beacons on the sphere surface with center the beacon itself and radius the distance measured. 3/4 of all particles are *PSOParticle*. The rest are *FollowBestParticle*.

```
/path/to/pylocating$ scripts/bestfitnessgraph.sh start_from_sphere_surface 16
```

arguments:

- 16: the environment contains 16 particles.

***note*: the number of particle should be divisible for 4 (the number of** beacons).

4.4 Benchmarks 1 - config 1

Evaluate distance error as a function of swarm size:

- error introduced: 3
- inertial weight: 1
- cognition: 2
- social: 2
- number of particles: range [10, 300]
- max particle velocity: 0.5
- interations per particle: 60

There is only one environmnent where all *PSOParticle* is connected.

The virtual space where the 4 beacons and the point is inserted is defined by:

- center: [1000, 1000, 1000]
- side length: 100

Every time the benchmark is started, their position are chosen randomly inside this cube. The distance error introduced is fixed and moved every time in a different position in the space. The benchmark is executed 100 times:

```
examples/benchmark_1.config1.sh 100
```

At the end of execution, you can see */tmp/benchmark_1.config1.jpg* file generated.

4.5 Benchmarks 1 - config 2

Evaluate distance error as a function of social parameter:

- error introduced: 3
- inertial weight: 1
- cognition: 2
- social: range [0, 10]
- number of particles: 100
- max particle velocity: 5
- interations per particle: 60

There is only one environmnent where all *PSOParticle* is connected.

The virtual space where the 4 beacons and the point is inserted is defined by:

- center: [1000, 1000, 1000]
- side length: 100

Every time the benchmark is started, their position are chosen randomly inside this cube. The distance error introduced is fixed and moved every time in a different position in the space. The benchmark is executed 100 times:

```
examples/benchmark_1.config2.sh 100
```

At the end of execution, you can see */tmp/benchmark_1.config2.jpg* file generated.

4.6 Benchmarks 2 - config 1

Evaluate distance error as a function of *FollowBestParticle* swarm size:

- error introduced: 3
- inertial weight: 1
- cognition: 2
- social: 2
- number of particles: range 80
- number of particles: range [1, 20]
- max particle velocity: 5
- interations per particle: 60

There are two environmnents:

- the first where *PSOParticle* are inserted
- the second where *FollowBestParticle* are inserted

The two environmnents are connected together.

The virtual space where the 4 beacons and the point is inserted is defined by:

- center: [1000, 1000, 1000]
- side length: 100

Every time the benchmark is started, their position are chosen randomly inside this cube. The distance error introduced is fixed and moved every time in a different position in the space. The benchmark is executed 100 times:

```
examples/benchmark_2.config1.sh 100
```

At the end of execution, you can see */tmp/benchmark_2.config1.jpg* file generated.

CHAPTER 5

Source

CHAPTER 6

Reference

6.1 pylocating

CHAPTER 7

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

7.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

7.2 Documentation improvements

pylocating could always use more documentation, whether as part of the official pylocating docs, in docstrings, or even on the web in blog posts, articles, and such.

7.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/hachreak/pylocating/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

7.4 Development

To set up *pylocating* for local development:

1. Fork pylocating on GitHub.
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/pylocating.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

7.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`) [1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

7.4.2 Tips

To run a subset of tests:

```
tox -e envname -- py.test -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 8

Authors

- Leonardo Rossi - http://hachreak.org

CHAPTER 9

Changelog

9.1 0.1.0 (2015-11-11)

- First release on PyPI.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search