OpenTNSim Documentation

Release 1.0.0

Mark van Koningsveld

Oct 13, 2021

Contents:

OpenTNSim is a python package for the investigation of traffic behaviour on networks to compare the consequences of different traffic scenarios and network configurations.

Welcome to OpenTNSim documentation! Please check the contents below for information on installation, getting started and actual example code. If you want to dive straight into the code you can check out our GitHub page or the working examples presented in Jupyter Notebooks.

CHAPTER 1

Installation

1.1 Stable release

To install OpenTNSim, run this command in your terminal:

```
# Use pip to install OpenTNSim
pip install opentnsim
```

This is the preferred method to install OpenTNSim, as it will always install the most recent stable release.

If you do not pip installed, this Python installation guide can guide you through the process.

1.2 From sources

The sources for OpenTNSim can be downloaded from the Github repo.

You can either clone the public repository:

```
# Use git to clone OpenTNSim
git clone git://github.com/TUDelft-CITG/OpenTNSim
```

Or download the tarball:

```
# Use curl to obtain the tarball
curl  -OL https://github.com/TUDelft-CITG/OpenTNSim/tarball/master
```

Once you have a copy of the source, you can install it with:

```
# Use python to install
python setup.py install
```

CHAPTER 2

OpenTNSim

This page lists all functions and classes available in the OpenTNSim.model and OpenTNSim.core modules. For examples on how to use these submodules please check out the Examples page, information on installing OpenCLSim can be found on the Installation page.

2.1 Submodules

The main components are the Model module and the Core module. All of their components are listed below.

2.2 opentnsim.model module

Vessel generator.

class opentnsim.model.**Simulation**(*simulation_start*, *graph*, *scenario=None*)

Bases: *opentnsim.core.Identifiable*

A class to generate vessels from a database

add_vessels(*origin*, *destination*, *vessel=None*, *vessel_generator=None*, *arrival_distribution=1*, *arrival_process='Markovian'*)

Make arrival process

environment: simpy environment arrival_distribution: specify the distribution from which vessels are generated, int or list arrival_process: process of arrivals

run(*duration=86400*)

Run the simulation

duration: specify the duration of the simulation in seconds

class opentnsim.model.**VesselGenerator**(*vessel_type*, *vessel_database*, *loaded=None*, *random_seed=3*)

Bases: object

A class to generate vessels from a database

arrival_process(*environment*, *origin*, *destination*, *arrival_distribution*, *scenario*, *arrival_process*)
Make arrival process

environment: simpy environment arrival_distribution: specify the distribution from which vessels are generated, int or list arrival_process: process of arrivals

generate(*environment*, *vessel_name*, *scenario=None*)
Generate a vessel

2.3 opentnsim.core module

Main module.

class opentnsim.core.**ConsumesEnergy**(*P_installed*, *L_w*, *C_b*, *nu=1e-06*, *rho=1000*, *g=9.81*, *x=2*, *eta_0=0.6*, *eta_r=1.0*, *eta_t=0.98*, *eta_g=0.96*, *c_stern=0*, *one_k2=2.5*, **args*, ***kwargs*)
Bases: object

Mixin class: Something that consumes energy.

P_installed: installed engine power [kW] L_w: weight class of the ship (depending on carrying capacity) (classes: L1 (=1), L2 (=2), L3 (=3)) C_b: block coefficient ('fullness') [-] nu: kinematic viscosity [m^2/s] rho: density of the surrounding water [kg/m^3] g: gravitational accelleration [m/s^2] x: number of propellors [-] eta_0: open water efficiency of propellor [-] eta_r: relative rotative efficiency [-] eta_t: transmission efficiency [-] eta_g: gearing efficiency [-] c_stern: determines shape of the afterbody [-] one_k2: appendage resistance factor [-] c_year: construction year of the engine [y]

calculate_appendage_resistance(*V_0*)

3) Appendage resistance

- 3rd resistance component defined by Holtrop and Mennen (1982)
- Appendages (like a rudder, shafts, skeg) result in additional frictional resistance

calculate_emission_factors_total()
Total emission factors:

- The total emission factors can be computed by multiplying the general emission factor by the correction factor

calculate_engine_age()
Calculating the construction year of the engine, dependend on a Weibull function with shape factor 'k', and scale factor 'lmb', which are determined by the weight class L_w

calculate_frictional_resistance(*V_0*, *h*)

1) Frictional resistance

- 1st resistance component defined by Holtrop and Mennen (1982)
- A modification to the original friction line is applied, based on literature of Zeng (2018), to account for shallow water effects

calculate_properties()
Calculate a number of basic vessel properties

calculate_residual_resistance(*V_0*, *h*)

5) Residual resistance terms

- Holtrop and Mennen (1982) defined three residual resistance terms:
- 1) Resistance due to the bulbous bow (not incorporated since inland ships in general don't have a bulb)
- 2) Resistance due to immersed transom
- 3) Resistance due to model-ship correlation

calculate_total_power_required()
Total required power:

- The total required power is the sum of the power for systems on board (P_hotel) + power required for propulsion (P_BHP)
- The P_BHP depends on the calculated resistance

calculate_total_resistance(*V_0*, *h*)
Total resistance:

The total resistance is the sum of all resistance components (Holtrop and Mennen, 1982)

calculate_viscous_resistance()

2) Viscous resistance

- 2nd resistance component defined by Holtrop and Mennen (1982)
- Form factor (1 + k1) has to be multiplied by the frictional resistance R_f, to account for the effect of viscosity

calculate_wave_resistance(*V_0*, *h*)

4) Wave resistance

- 4th resistance component defined by Holtrop and Mennen (1982)
- When the speed or the vessel size increases, the wave making resistance increases
- In shallow water, the wave resistance shows an asymptotical behaviour by reaching the critical speed

correction_factors()
Correction factors:

- The correction factors have to be multiplied by the general emission factors, to get the total emission factors
- The correction factor takes into account the effect of the partial engine load
- When the partial engine load is low, the correction factors are higher (engine is less efficient)
- Based on literature TNO (2019)

emission_factors_general()
General emission factors:

This function computes general emission factors, based on construction year of the engine. - Based on literature TNO (2019)

Please note: later on a correction factor has to be applied to get the total emission factor

karpov(*V_0*, *h*)
Intermediate calculation: Karpov

- The Karpov method computes a velocity correction that accounts for limited water depth (corrected velocity V2)
- V2 has to be implemented in the wave resistance and the residual resistance terms

class opentnsim.core.**ContainerDependentMovable**(*compute_v*, **args*, ***kwargs*)
Bases: *opentnsim.core.Movable*, *opentnsim.core.HasContainer*

ContainerDependentMovable class Used for objects that move with a speed dependent on the container level compute_v: a function, given the fraction the container is filled (in [0,1]), returns the current speed

current_speed

class opentnsim.core.**HasContainer**(*capacity*, *level=0*, *total_requested=0*, **args*, ***kwargs*)
Bases: *opentnsim.core.SimpyObject*

Mixin class: Something with a storage capacity capacity: amount the container can hold level: amount the container holds initially container: a simpy object that can hold stuff total_requested: a counter that helps to prevent over requesting

filling_degree

is_loaded

class opentnsim.core.**HasLength**(*length*, *remaining_length=0*, *total_requested=0*, **args*, ***kwargs*)
Bases: *opentnsim.core.SimpyObject*

Mixin class: Something with a storage capacity capacity: amount the container can hold level: amount the container holds initially total_requested: a counter that helps to prevent over requesting

class opentnsim.core.**HasLockDoors**(*node_1*, *node_3*, **args*, ***kwargs*)
Bases: *opentnsim.core.SimpyObject*

class opentnsim.core.**HasResource**(*nr_resources=1*, *priority=False*, **args*, ***kwargs*)
Bases: *opentnsim.core.SimpyObject*

Something that has a resource limitation, a resource request must be granted before the object can be used.

nr_resources: nr of requests that can be handled simultaneously

class opentnsim.core.**Identifiable**(*name*, *id=None*, **args*, ***kwargs*)
Bases: object

Mixin class: Something that has a name and id

name: a name id: a unique id generated with uuid

class opentnsim.core.**IsLock**(*node_1*, *node_2*, *node_3*, *lock_length*, *lock_width*, *lock_depth*, *doors_open*, *doors_close*, *wlev_dif*, *disch_coeff*, *grav_acc*, *opening_area*, *opening_depth*, *simulation_start*, *operating_time*, **args*, ***kwargs*)
Bases: *opentnsim.core.HasResource*, *opentnsim.core.HasLength*, *opentnsim.core.HasLockDoors*, *opentnsim.core.Identifiable*, *opentnsim.core.Log*

Mixin class: Something has lock object properties properties in meters operation in seconds

change_water_level(*side*)
Change water level and priorities in queue

convert_chamber(*environment*, *new_level*, *number_of_vessels*)
Convert the water level

operation_time(*environment*)

class opentnsim.core.**IsLockLineUpArea**(*node*, *lineup_length*, **args*, ***kwargs*)
Bases: *opentnsim.core.HasResource*, *opentnsim.core.HasLength*, *opentnsim.core.Identifiable*, *opentnsim.core.Log*

Mixin class: Something has lock object properties properties in meters operation in seconds

class opentnsim.core.**IsLockWaitingArea**(*node*, **args*, ***kwargs*)
Bases: *opentnsim.core.HasResource*, *opentnsim.core.Identifiable*, *opentnsim.core.Log*

Mixin class: Something has lock object properties properties in meters operation in seconds

class opentnsim.core.**Locatable**(*geometry*, **args*, ***kwargs*)
Bases: object

Mixin class: Something with a geometry (geojson format)

geometry: can be a point as well as a polygon

class opentnsim.core.**Log**(**args*, ***kwargs*)
Bases: *opentnsim.core.SimpyObject*

Mixin class: Something that has logging capability

log: log message [format: 'start activity' or 'stop activity'] t: timestamp value: a value can be logged as well geometry: value from locatable (lat, lon)

get_log_as_json()

log_entry(*log*, *t*, *value*, *geometry_log*)
Log

class opentnsim.core.**Movable**(*v=4*, **args*, ***kwargs*)
Bases: *opentnsim.core.Locatable*, *opentnsim.core.Routeable*, *opentnsim.core.Log*

Mixin class: Something can move

Used for object that can move with a fixed speed

geometry: point used to track its current location v: speed

current_speed

move()
determine distance between origin and destination, and yield the time it takes to travel it Assumption is that self.path is in the right order - vessel moves from route[0] to route[-1].

pass_edge(*origin*, *destination*)

class opentnsim.core.**Neighbours**
Bases: object

Can be added to a locatable object (list) travel_to: list of locatables to which can be travelled

class opentnsim.core.**Routeable**(*route*, *complete_path=None*, **args*, ***kwargs*)
Bases: object

Mixin class: Something with a route (networkx format)

route: a networkx path

class opentnsim.core.**SimpyObject**(*env*, **args*, ***kwargs*)
Bases: object

General object which can be extended by any class requiring a simpy environment env: a simpy Environment

class opentnsim.core.**VesselProperties**(*type*, *B*, *L*, *H_e*, *H_f*, *T_e*, *T_f*, **args*, ***kwargs*)
Bases: object

Mixin class: Something that has vessel properties This mixin is updated to better accommodate the ConsumesEnergy mixin

type: can contain info on vessel type (avv class, cemt_class or other) B: vessel width L: vessel length H_e: vessel height unloaded H_f: vessel height loaded T_e: draught unloaded T_f: draught loaded

Add information on possible restrictions to the vessels, i.e. height, width, etc.

H
Calculate current height based on filling degree

T
Calculate current draught based on filling degree

Here we should implement the rules from Van Dorsser et al https://www.researchgate.net/publication/344340126_The_effect_of_low_water_on_loading_capacity_of_inland_ships

get_route(*origin*, *destination*, *graph=None*, *minWidth=None*, *minHeight=None*, *minDepth=None*, *randomSeed=4*)
Calculate a path based on vessel restrictions

2.4 Module contents

CHAPTER 3

Contributing

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

You can contribute in many ways:

3.1 Types of Contributions

3.1.1 Report Bugs

Report bugs at https://github.com/TUDelft-CITG/OpenTNSim/issues.

If you are 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.

3.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

3.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

3.1.4 Write Documentation

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

3.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/TUDelft-CITG/OpenTNSim/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 :)

3.2 Get Started!

Ready to contribute? Here's how to set up *OpenTNSim* for local development.

1. Fork the *OpenTNSim* repository on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/OpenTNSim.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv opentnsim
$ cd opentnsim/
$ python setup.py develop
```

4. Create a branch for local development:

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

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 opentnsim tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. The style of OpenTNSim is according to Black. Format your code using Black witht the following lines of code:

```
$ black opentnsim
$ black tests
```

   You can install black using pip.

7. 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
```

8. Submit a pull request through the GitHub website.

3.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.4, 3.5 and 3.6, and for PyPy. Check CircleCI and make sure that the tests pass for all supported Python versions.

3.4 Tips

To run a subset of tests:

```
$ py.test tests.test_opentnsim
```

To make the documentation pages $ make docs # for linux/osx

For windows $ del docsopentnsim.rst $ del docsmodules.rst $ sphinx-apidoc -o docs/ opentnsim $ cd docs $ make html $ start explorer _buildhtmlindex.html

3.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 4

Credits

4.1 Development Lead

- Mark van Koningsveld
- Joris den Uijl

4.2 Contributors

Various MSc projects

- Jeroen van der Does de Willebois, 2019. **Assessing the impact of quay-wall renovations on the nautical traffic in Amsterdam: A simulation study.** MSc thesis. Delft University of Technology, Civil Engineering and Geosciences, Hydraulic Engineering - Ports and Waterways. Delft, the Netherlands.
- Leonore Vehmeijer, 2019. **Measures for the reduction of CO2 emissions, by the inland shipping fleet, on the Rotterdam-Antwerp corridor.** MSc thesis. Delft University of Technology, Civil Engineering and Geosciences, Hydraulic Engineering - Ports and Waterways. Delft, the Netherlands.
- Sophie Ensing, 2019. **Agent-based modeling and simulation of public transport to identify effects of network changes on passenger flows.** MSc thesis. University of Amsterdam, Faculty of Science, Data Science, Information Studies. Amsterdam, the Netherlands.

CHAPTER 5

History

5.1 1.0.0 (2020-05-07)

- Release in preparation of SmartShipping project

5.2 0.1.0 (2019-07-18)

- First release to PyPI and rename to OpenTNSim

CHAPTER 6

Version conventions

This package is being developed continuously. Branch protection is turned on for the master branch. Useful new features and bugfixes can be developed in a separate branch or fork. Pull requests can be made to integrate updates into the master branch. To keep track of versions, every change to the master branch will receive a version tag. This page outlines the version tags' naming convention.

Each change to the master branch is stamped with a unique version identifier. We use sequence based version identifiers, that consist of a sequence of three numbers: the first number is a major change identifier, followed by a minor change idenfier and finally a maintenance identifier. This leads to version identifiers of the form:

major.minor.maintenance (example: 1.2.2)

The following guideline gives an idea what types of changes are considered major changes, minor changes and maintenance:

- Major changes (typically breaking changes) -> major + 1
- Minor changes (typically adding of new features) -> minor + 1
- Maintenance (typically bug fixes and updates in documentation -> maintenance + 1

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index