JÜLICH PEDESTRIAN SIMULATOR

FORSCHUNGSZENTRUM JÜLICH GMBH

User's Guide Version 0.8.2

Stand: October 30, 2017

Contents

1 DISCLAIMER **3**

2 Introduction **4**

2.1 Requirements 7

3 JPScore **10**

3.1 File formats 10

3.1.1 Inifile 10

3.1.2 Geometry file 17

3.1.3 Trajectory file 23

3.2 Models 26

3.3 Direction strategies 35

3.4 Route finding 37

3.5 Validation and Verification 39

3.5.1 RiMEA tests 39

3.5.2 Jülich tests 42

4 JPSreport **48**

4.1 File formats 48

4.1.1 Inifile 48

4.1.2 Trajectory file 55

4.2 Measurement methods 58

4.2.1 Method A 58

4.2.2 Method B 59

4.2.3 Method C 60

4.2.4 Method D 61

5 Publications **62**

5.1 Publications 62

1 DISCLAIMER

In no event shall JuPedSim be liable to any party for direct, indirect, special, incidental, or consequential damages, including lost profits, arising out of the use of this software and its documentation, even if JuPedSim has been advised of the possibility of such damage.

JuPedSim specifically disclaims any warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The software and accompanying documentation, if any, provided hereunder is provided “as is”. JuPedSim has no obligation to provide maintenance, support, updates, enhancements, or modifications.

2 Introduction

Open source framework for simulating, analyzing and visualizing pedestrian dynamics.

Goal

The primary goal of JuPedSim is to provide students and researchers with a framework to investigate pedestrian dynamics
and focus on research, i.e. development and validation of new models or model features, analysis of experiments and proper visualization of results.

JuPedSim is currently focusing on evacuation, but is easily extendable to cover other areas
e.g. passengers exchange, commuter traffic in railway stations etc.

Organization of the code

JuPedSim consists of four modules which are loosely coupled and can be used independently at the moment. These are:

1. jpscore: the core module computing the trajectories. See list of implemented models.

2. jpsreport: a tool for analyzing the trajectories and validating the model.
   It implements a couple of measurement methods including the Voronoi-method for calculating the density.

3. jpsvis: a tool for visualizing the input (geometry) and output (trajectories) data.

4. jpseditor: a tool for creating and editing geometry files with dxf import/export capabilities.

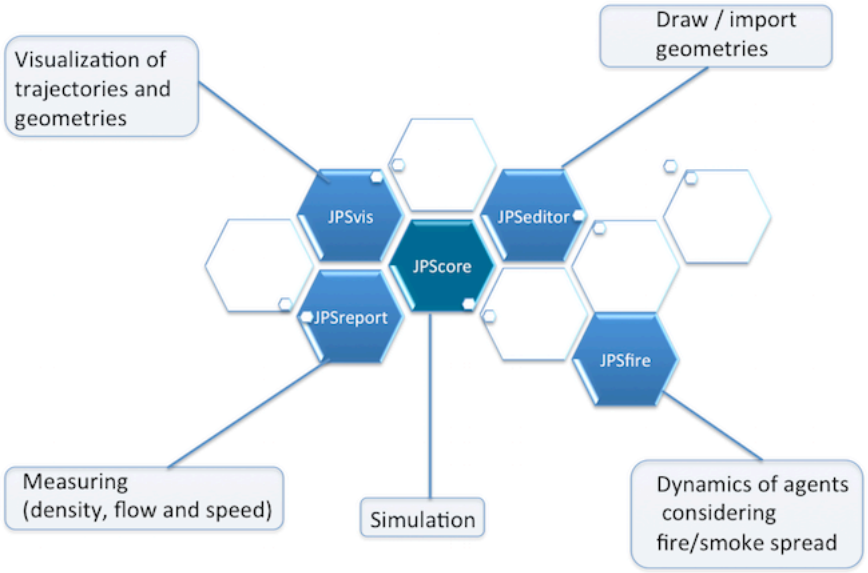

Fig. 2.1: Structure of JuPedSim: interacting modules

Showcase and tutorials

To highlight some features of `JuPedSim` we have uploaded some videos and tutorials on
our YouTube channel.

Quick Install

The recommended method to get `JuPedSim` is to download the code and compile it, following the following instructions:

```
git clone https://github.com/JuPedSim/JuPedSim.git
cd JuPedSim
git submodule update --init --recursive
make -f Makefile.cmake check
```

Note: it is very important to update the submodule, otherwise the they will be empty!

`CMake` will eventually report any missing packages. Please fix these errors/warnings first **before** proceeding with the compilation of `JuPedSim` with

```
make -f Makefile.cmake check release
```

Support

We are heavily working on this project which means that:

- It's not done. We will be releasing new enhancements, bug fixes etc.
- We love your support. If you find any errors or have suggestions, please write an issue in our issue-tracker.
  We will try hard to fix it.
- Be patient. We are scientists and PhD/master students. Therefore, we primarily care about our research and theses.

Contact/Feedback

You have found a bug in the code? Please help us with your feedback. You can contact us per mail at
`info at jupedsim dot org`.

2.1 Requirements

In order to compile JuPedSim it is necessary to first install the required libraries.

Supported compilers

Any compiler with support for C++11.

Tested with

- g++ (linux/osx)
- clang (osx)
- Visual Studio 2013 (Windows)

Required tools

- cmake: see this tutorial for a brief overview.
- Python (highly recommended): needed to plot results of analysis.

Required libraries

- Boost library: necessary for jpscore and jpsreport

Install Boost (at least v1.59)

Mac

For brew users:

```
brew install boost
```

And for port users

```
sudo port install boost
```

Linux

You can compile boost using the following snippet:

```
boost_version=1.61.0
boost_dir=boost_1_61_0

wget http://downloads.sourceforge.net/project/boost/boost/${
    boost_version}/${boost_dir}.tar.gz
tar xfz ${boost_dir}.tar.gz
rm ${boost_dir}.tar.gz
cd ${boost_dir}
./bootstrap.sh --with-libraries=filesystem,test,system
sudo ./b2 --without-python --prefix=/usr -j 4 link=shared runtime-
    link=shared install
cd ..
rm -rf ${boost_dir}
sudo ldconfig
```

(download this snippet as a script).

Note: Debian's and Ubuntu's install manager offer an old version of Boost, which is not supported by `JuPedSim`.

Windows

```
bootstrap
b2  variant=release --build-type=complete
```

See also Getting started on Windows.

This script can be useful, in case you are using Visual Studio.

Download it and put it in the same directory as Boost. Depending on your Boost version and VS,
you may want to adapt in the script the variables `boost_dir` and `msvcver`.

Test Boost installation

You can test your Boost installation by using this minimal example.

3 JPScore

3.1 File formats

`jpscore` needs as input an “inifile” and a geometry file. B successful simulation it produces a trajectory file.

3.1.1 Inifile

With the `inifile` the simulation with `jpscore` can be controlled.

The typical structure of an `inifile` is as follows:

```
    <header>
         <!-- seed , geometry, output format -->
    </header>

    <traffic_constraints>
        <!-- traffic information: e.g closed doors or smoked rooms --
    >
    </traffic_constraints>

    <goals>
    <!-- goals (closed polygons) outside the geometry-->
    </goals>

    <agents>
        <agents_distribution>
            <!--persons information and distribution -->
        </agents_distribution>
    </agents>

    <operational_models>
        <model id="n" description="name">
            <!-- parameters of model (<n>, "name")  -->
        </model>
        <!-- other models can be defined -->
    </operational_models>

    <route_choice_models>
        <router router_id="n" description="name">
```

```
            <!-- parameters of router (<n>, "name")  -->
        </router>
        <!-- other routers can be defined -->
    </route_choice_models>

</JuPedSim>
```

The main components of the inifile are:

- header
- traffic constraints
- goals
- agents
- operational models.
- route choice models.

Header

The header comprises the following elements:

- `<seed>s</seed>`

  Set the `seed` value of the random number generator to `s`. If missing the current time (`time(NULL)`), is used i.e. random initial conditions.

- `<max_sim_time>t</max_sim_time>`
  the maximal simulation time in seconds.

- `<num_threads>n</num_threads>` the number of used cores.

- `<show_statistics>true</show_statistics>` Show different aggregate statistics e.g. the usage of the doors. (default: false)

- `<logfile>log.txt</logfile>` save relevant information about the simulation to a log file.
  Useful to keep track of warnings or errors that may rise during a simulation.

- The trajectory file

```
<trajectories format="xml-plain" fps="8" color_mode="velocity">  <
    file location="trajectories.xml" />  </trajectories>
```

The options for the format are

- `xml-plain`: the default xml format. It can lead to large files. See section xml-plain.

- `plain`: simple text format. See section plain-text.

- The value `fps` defines the frame rate per second for the trajectories.

  - `color_mode`: coloring agents in the trajectories. Options are:

    * `velocity` (default): color is proportional to speed (slow –> red).

    * `spotlight`

    * `group`: color by group

    * `knowledge`

    * `router`

    * `final_goal`

    * `intermediate_goal`

  - `file location` defines the location of the trajectories.
    All paths are relative to the location of the project file.

- `<geometry>geometry.xml</geometry>`
  The name and location of the geometry file. All file locations are relative to the actual location of the project file. See specification of the geometry format.

Traffic constraints

This section defines constraints related to the traffic.
At the moment the state of the doors can be changed (`open` or `close`)

```
<traffic_constraints>
    <!-- doors states are: close or open -->
    <doors>
        <door trans_id="4" caption="Main-gate" state="open" />
        <door trans_id="6" caption="Rear-gate" state="close" />
    </doors>
</traffic_constraints>
```

- `trans_id`: unique id of that specific door as defined in the geometry file. See geometry.
- `caption`: optional parameter defining the caption of the door.
- `state` defines the state of the door. Options are `close` or `open`.

Goals

Additional goals might be defined **outside** the geometry.
They should *NOT* overlap with any walls or be inside rooms.
It is recommended to position them near the exits.

Goals are defined with close polygons, with *the last vertex is equal to the first one*.

```
<routing>
    <goals>
        <goal id="0" final="false" caption="goal 1">
            <polygon>
                <vertex px="-5.0" py="-5.0" />
                <vertex px="-5.0" py="-2.0" />
                <vertex px="-3.0" py="-2.0" />
                <vertex px="-3.0" py="-5.0" />
```

```
                <vertex px="-5.0" py="-5.0" />
                </polygon>
        </goal>
        <goal id="1" final="false" caption="goal 2">
            <polygon>
                <vertex px="15.0" py="-5.0" />
                <vertex px="17.0" py="-5.0" />
                <vertex px="17.0" py="-7.0" />
                <vertex px="15.0" py="-7.0" />
                <vertex px="15.0" py="-5.0" />
            </polygon>
        </goal>
    </goals>
</routing>
```

Agents

There are two ways to distribute agents for a simulation:

- random distribution in a specific area *before* the simulation starts.
- distribution by means of sources *during* the simulation.

Agents_distribution An example how to define agent's characteristics with different number of
attributes is as follows

```
    <agents>
        <agents_distribution>
            <group group_id="1" room_id="0" number="10" />

            <group group_id="2" room_id="0" subroom_id="0" number="10
    "
                goal_id="" router_id="1" />

        </agents_distribution>
    </agents>
```

- `group_id`: mandatory parameter defining the unique id of that group.
- `number`: mandatory parameter defining the number of agents to distribute.

- `room_id`: mandatory parameter defining the room where the agents should be
  randomly distributed.

- `subroom_id`: defines the id of the subroom where the agents should be distributed.
  If omitted then the agents are homogeneously distributed in the room.

- `goal_id`: should be one of the `id`s defined in the section goals.
  If omitted or is `-1` then the shortest exit to the outside is chosen by the agent.

- `router_id`: defines the route choice model to be used. See documentation of available routers.

- `age`: not yet used by the operational models.

- `gender`: not yet used.

- `height`: not yet used.

- `patience`: this parameter influences the route choice behavior when using the quickest path router.
  It basically defines how long a pedestrian stays in jams before attempting a rerouting.

- `pre_movement_mean` and `pre_movement_sigma`: premovement time is Gauss-distributed $\mathcal{N}(\mu, \sigma^2)$.

- Risk tolerance can be Gauss-distributed, or beta-distributed.
  If not specified then it is defined as $\mathcal{N}(1, 0)$:

  - `risk_tolerance_mean` and `risk_tolerance_sigma`: $\mathcal{N}(\mu, \sigma^2)$.

  - `risk_tolerance_alpha` and `risk_tolerance_beta`: $Beta(\alpha, \beta)$.

- `x_min`, `x_max`, `y_min` and `y_max`: define a bounding box where agents should be distributed.

- `startX`, `startY`: define the initial coordinate of the agents. This might be useful for testing/debugging.
  Note that these two options are only considered if `number=1`.

- `agent_parameter_id`: choose a set of parameters for the operational models.

Sources Besides distributing agents randomly before the simulation starts, it is possible to define sources in order to "inject"
new agents in the system during the simulation.

```
<agents_sources>
<source id="1" frequency="2", agents_max="10" group_id="1", caption="
    caption" greedy="true"/>
</agents_sources>
```

- `id`: id of the source
- `frequency`: number of pedestrians per second.
- `agents_max`: maximal number of agents produced by that source.
- `group_id`: group id of the agents. This `id` should match a predefined group in the section Agents_distribution.
- `caption`: caption
- `greedy` (default `false`): returns a Voronoi vertex randomly with respect to weights proportional to squared distances.
  For vertexes v_i and distances d_i to their surrounding seeds
  calculate the probabilities p_i as

$$p_i = \frac{d_i^2}{\sum_j^n d_j^2}.$$

  If this attribute is set to `true`, the greedy approach is used.
  That means new agents will be placed on the vertex with the biggest distance to the surrounding seeds.

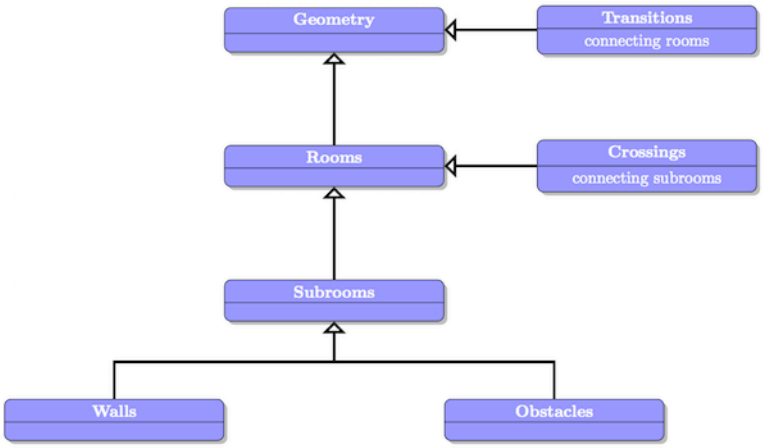

Fig. 3.1: The main components defining a geometry

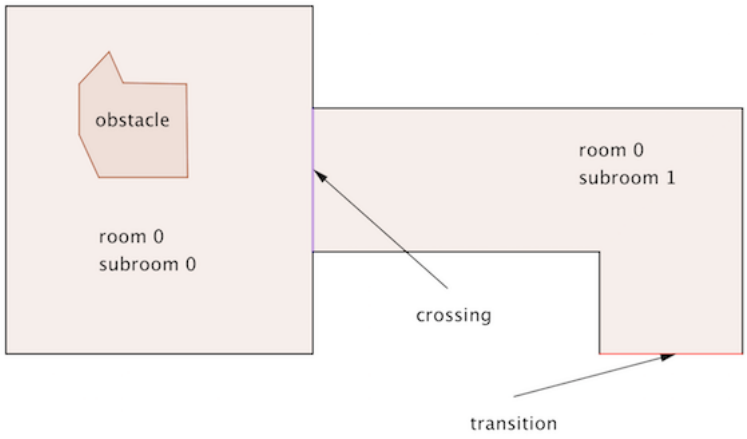

Fig. 3.2: A corner defined by one room and two subrooms

3.1.2 Geometry file

The main structure of the geometry file is as follows

Following picture shows a sample geometry with one room, two subrooms and one obstacle.

`subrooms` and `rooms` are two different concepts to organize a geometry.
Basically, it is up to the user to organize its geometry “roomwise” or “subroom-wise”.

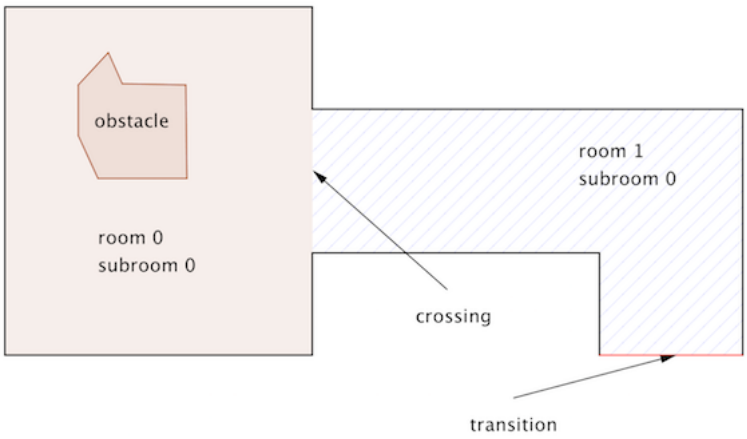

Fig. 3.3: A corner defined by two different rooms

For instance the above mentioned geometry could also be organized using two rooms as follows:

A geometry can be produces manually (for small scenarios) or with jpseditor.
In both cases it is recommended to visualize the geometry once finished with jpsvis.

The main components of a geometry are:

- Rooms
- Subrooms
- Obstacles
- Transitions
- Crossings

Rooms

The geometry contains at least one room and one transition.
Each room has a unique `id`, an optional caption and at least one `subroom`.
Two rooms are separated by either walls or transitions.

```
<rooms>
    <room id="0" caption="hall" >
</rooms>
```

Subrooms

Subrooms define the navigation mesh, i.e the walkable areas in the geometry.
Each `subroom` is bounded by at least one `crossing`.
Here a sample:

```
<subroom id="1" class="stair" A_x="1.2" B_y="0" C="0">
    <polygon caption="wall">
        <vertex px="0.0" py="1.0"/>
        <vertex px="5.0" py="1.0"/>
    </polygon>
    <polygon caption="wall">
        <vertex px="0.0" py="3.0"/>
        <vertex px="5.0" py="3.0"/>
    </polygon>
    <up px="5.0" py="2"/>
    <down px="0.0" py="2"/>
</subroom>
```

- `id` mandatory parameter, also referred by crossings and transitions.
- `class` optional parameter defining the type of the subroom. At the moment two classes are defined:
  - `floor`
  - `stairs` take additionally
    `<up px="-5.0" py="2" />` and `<down px="0.0" py="2"/>`, which are used for visualisation purposes.
- A_x, B_y, and C are optional parameter for the explicit plane equation of the subroom,
  for the construction of a 3D environment and should be used to describe stairs.
  The plane equation is given by: $Z = Ax + By + C$.

For instance, if the stair goes through the following points:
$P_1(1,0,0)$, $P_2(0,1,0)$ and $P_3(0,0,1)$
then the equation is given by: $Z = -x - y + 1$.

- `polygon` describes the walls as a sequence of vertexes.

To ease navigation, it is recommended to always use convex subrooms.
In the case the subroom is not convex, additional navigation lines might be required or the floor field router should be used.

Obstacles

One or more obstacles can also be defined within a `subroom`.

Sample obstacle in a subroom

```
<obstacle id="0" caption="table" height="1.0" >
    <polygon>
      <vertex px="12.0" py="6.0"/>
    <vertex px="13.0" py="6.0"/>
    <vertex px="13.0" py="5.5"/>
    <vertex px="12.0" py="5.5"/>
    <vertex px="12.0" py="6.0"/>
    </polygon>
</obstacle>
```

- `id`, mandatory unique identifier for this obstacle.
- `caption`, used in the visualisation.
- `height`, optional parameter, not used at the moment
- `polygon`, describing the obstacle as a sequence of vertex.

Transitions

A `transition` defines the connection between two rooms and is basically a door.
It can be close or open (see "traffic constraints").

An example transition between two rooms

```
<!-- exits between rooms or to outside (room with index = -1) -->
<transition id="1" caption="main exit" type="emergency"
room1_id="0" subroom1_id="1" room2_id="-1" subroom2_id="-1">
    <vertex px="15.0" py="-5.0"/>
    <vertex px="17.0" py="-5.0"/>
</transition>
```

- `id`, mandatory unique identifier.
  The `id` is also used to close or open the door in the "traffic constraints" section of the inifile.

- `caption`, optional, used in the visualisation.

- `type`, optional.

- `room1_id`, the first room sharing this transition. The order is not important.

- `subroom1_id`, the first `subroom` located in `room_1`.

- `room2_id`, the second room sharing this transition.
  The order is not important.
  If there is no second room (meaning this transition is connected to the outside), then use `-1`.

- `subroom2_id`, the second `subroom` sharing this transition. The order is not important.
  If there is no second `subroom` (meaning this transition is connected to the outside), then use `-1`.

- `vertex`: define two ending points of the `transition`.

Crossings

A `crossing` defines the connection between two subrooms inside the same room. Unlike `transition`, they are always open.

A sample `crossing` between two subrooms

```
<!-- virtual exits between subrooms -->
<crossing id="0" subroom1_id="0" subroom2_id="1">
  <vertex px="10.0" py="6.0"/>
  <vertex px="10.0" py="4.0"/>
</crossing>
```

- `id`, mandatory unique identifier for this `crossing`.
- `subroom1_id`, the first `subroom`
- `subroom2_id`, the second `subroom` sharing this transition. The order is not important.
  If there is no second subroom (meaning this transition is connected to the outside),
  then use `-1`.
- `vertex`: define two ending points of the `crossing`.

3.1.3 Trajectory file

The results of the simulation are written to files or streamed to a network socket.

Possible formats are:

- `xml-plain` which is the default xml format
- `plain` a flat format (just numbers)

Note that if you are using the streaming mode or want to visualize the trajectories with `jpsvis`,
the format should be `xml-plain`.

xml-plain

The file has three main sections: `header`, `geometry` and `frames`.

```
<header version = "0.8">
    <agents>1</agents>
  <frameRate>8</frameRate>
</header>
```

where

- `agents`: The total number of agents at the beginning of the simulation.
- `frameRate`: Divide the total number of frames by the framerate to obtain the overall evacuation time.

The geometry can be completely embedded within the trajectories or a reference to a file can be supplied.

```
<geometry>
  <file location="corridor_geometry.xml"/>
</geometry>
```

The coordinates of the trajectory are defined in the session `frames`

```
<frame ID="0">
    <agent ID="1" x="660.00"  y="333.00"  z="30.00"
  rA="17.94"  rB="24.94"  e0="-168.61" eC="0"/>
</frame>
<frame ID="1">
<agent ID="1" x="658.20"  y="332.86"  z="30.00"
rA="31.29"  rB="23.87"  e0="-175.41" eC="54"/>
</frame>
```

- `ID` the id of the pedestrians starting with 1.
- `x, y, z` the position of the agent.
- `rA, rB` The shape which is defined by a circle (ellipse) drawn around a human-like figure.
  `radiusA` and `radiusB` are respectively the semi major axis and the semi minor axis of the ellipse,
  if the modeled pedestrians' shape is an ellipse.
  Otherwise, if it is a circle those values should be equal to the radius of the circle.
- `e0, eC` are the "ellipseOrientation" and the "ellipseColor".
  "ellipseOrientation" is the angle between the major axis and the X-axis (zero for circle).
  A color can also be provided, for example for displaying change in velocity.
  The colours are in the range `[0=red, 255=green]` and define the proportion between
  the desired speed (v_0) and the instantaneous velocity.

A sample trajectory in the xml format is

```
<?xml version="1.0" encoding="UTF-8"?>
<trajectories>
  <header version = "0.5">
    <agents>1</agents>
    <frameRate>8</frameRate>
  </header>

  <geometry>
    <file location="corridor_geometry.xml"/>
  </geometry>

  <frame ID="0">
```

```
    <agent ID="1" x="660.00"  y="333.00"  z="30.00"
    rA="17.94"  rB="24.94"  e0="-168.61"  eC="0"/>
  </frame>

  <frame ID="1">
    <agent ID="1" x="658.20"  y="332.86"  z="30.00"
  rA="31.29"  rB="23.87"  e0="-175.41"  eC="54"/>
  </frame>
</trajectories>
```

plain-text

The other format of the trajectory file is `plain-text`

A sample trajectory in the plain format is as follows:

```
        #description: simulation
        #framerate: 16
        #geometry: /home/sim/corridor.xml
        #ID: the agent ID
        #FR: the current frame
        #X,Y,Z: the agents coordinates in metres

        #ID FR  X   Y   Z
        1 0 28.21 131.57  0.00
        2 0 38.41 133.42  0.00
        1 1 28.21 131.57  0.00
        2 1 38.41 133.42  0.00
        1 2 28.24 131.57  0.00
        2 2 38.44 133.42  0.00
        1 3 28.29 131.57  0.00
        2 3 38.49 133.42  0.00
        1 4 28.36 131.57  0.00
        2 4 38.56 133.42  0.00
        1 5 28.44 131.57  0.00
        2 5 38.64 133.42  0.00
        1 6 28.54 131.57  0.00
        2 6 38.74 133.42  0.00
        1 7 28.65 131.57  0.00
        2 7 38.85 133.42  0.00
        1 8 28.77 131.57  0.00
```

3.2 Models

Several operational models are implemented in `jpscore`.
An operational model defines how pedestrians moves from one time step to the next.
In the definition of agent's properties it is mandatory to precise the number of the model to be used e.g.:

```
<agents operational_model_id="n">
```

where `n` is 1, 2, 3, 4 or 5.

General model parameters (for all models)

The definition of any model parameter is composed of two different sections:

- **model_parameters**: Model specific parameter. See below in the different model sections.
- **agent_parameters**: These parameter are mainly specific for the shape of pedestrians
  or other pedestrian properties like desired speed, reaction time etc.

Model_parameters

- `<solver>euler</solver>`
  - The solver for the ODE. Only *Euler*. No other options.
- `<stepsize>0.001</stepsize>`:
  - The time step for the solver. This should be choosed with care. For force-based model it is recommended to take a value between 10^{-2} and 10^{-3} s.
    For first-order models, a value of 0.05 s should be OK.
    A larger time step leads to faster simulations, however it is too risky and

can lead to
numerical instability, collisions and overlapping among pedestrians.

  - Unit: s

- `<periodic>0</periodic>`

  - Set to 1 if a system with closed boundary conditions should be simulated. Default setting is 0.

  - This option is only implemented in *Tordeux2015* and is very geometry-specific (only for corridors) with predefined settings. See Utest/Validation/1test_1D/ for a use case.

- `<exit_crossing_strategy>3</exit_crossing_strategy>`

  - Positive values in $[1, 9]$. See Direction strategies for the definition of the strategies.

- `<linkedcells enabled="true" cell_size="2"/>`

  - Defines the size of the cells. This is important to get the neighbors of a pedestrians, which
    are all pedestrians within the eight neighboring cells. Larger cells, lead so slower simulations, since
    more pedestrian-pedestrian interactions need to be calculated.

  - Unit: m

Agent_parameters

The *agent parameters* are mostly identical for all models. Exceptions will be mentioned explicitly.

The parameters that can be specified in this section are Gauss distributed (default value are given).

Desired speed

- `<v0 mu="1.2" sigma="0.0" />`
  - Desired speed
  - Unit: m/s
- `<v0_upstairs mu="0.6" sigma="0.167" />`
  - Desired speed upstairs
  - Unit: m/s
- `<v0_downstairs mu="0.6" sigma="0.188" />`
  - Desired speed downstairs
  - Unit: m/s
- `<v0_idle_escalator_upstairs mu="0.6" sigma="0.0" />`
  - Speed of idle escalators upstairs
  - Unit: m/s
- `<v0_idle_escalator_downstairs mu="0.6" sigma="0.0" />`
  - Speed of idle escalators downstairs
  - Unit: m/s

Shape of pedestrians Pedestrians are modeled as ellipses with two semi-axes: a and b, where

$$a = a_{min} + a_\tau v,$$

and

$$b = b_{max} - (b_{max} - b_{min})\frac{v}{v^0}.$$

v is the peed of a pedestrian.

- `<bmax mu="0.15" sigma="0.0" />`
  - Maximal length of the shoulder semi-axis
  - Unit: m
- `<bmin mu="0.15" sigma="0.0" />`
  - Minimal length of the shoulder semi-axis
  - Unit: m
- `<amin mu="0.15" sigma="0.0" />`
  - Minimal length of the movement semi-axis. This is the case when $v = 0$.
  - Unit: m
- `<atau mu="0." sigma="0.0" />`
  - (Linear) speed-dependency of the movement semi-axis
  - Unit: s

Other parameters in this section are:

- `<tau mu="0.5" sigma="0.0" />`
  - Reaction time. This constant is used in the driving force of the force-based forces. Small $\rightarrow$ instantaneous acceleration.
  - Unit: s
- `<T mu="1" sigma="0.0" />`
  - Specific parameter for model 3 (Tordeux2015). Defines the slope of the speed function.

Generalized Centrifugal Force Model

Generalized Centrifugal Force Model is a force-based model.

Usage:

```
<model operational_model_id="1" description="gcfm">
```

Gompertz model

Gompertz Model is a force-based model.

Usage:

```
<model operational_model_id="2" description="gompertz">
```

Collision-free speed model

Collision-free speed model is a velocity-based model. See also this talk for more details about the model.

Usage:

```
<model operational_model_id="3" description="Tordeux2015">
```

Model parameters Besides the options defined in Model_parameters the following options are necessary for this model:

- `<force_ped a="5" D="0.2"/>`
  - The influence of other pedestrians is triggered by a and D where a is the strength if the interaction and D gives its range. The naming may be misleading, since the model is **not** force-based, but velocity-based.
  - Unit: m

- `<force_wall a="5" D="0.02"/>`:
  - The influence of walls is triggered by a and D where a is the strength if the interaction and D gives its range. A larger value of D may lead to blockades, especially when passing narrow bottlenecks.
  - Unit: m

The names of the aforementioned parameters might be misleading, since the model is *not* force-based. The naming will be changed in the future.

Agent parameters (recommendations) Actually, this model assumes circular pedestrian's shape, therefore the parameter for the semi-axes Agent_parameters should be chosen, such that circles with constant radius can be obtained.
For example:

```
<bmax mu="0.15" sigma="0.0" />
<bmin mu="0.15" sigma="0.0" />
<amin mu="0.15" sigma="0.0" />
<atau mu="0." sigma="0.0" />
```

This defines circles with radius 15 cm.

In summary the relevant section for this model could look like:

```
<model operational_model_id="3" description="Tordeux2015">
    <model_parameters>
        <solver>euler</solver>
        <stepsize>0.05</stepsize>
        <exit_crossing_strategy>3</exit_crossing_strategy>
        <linkedcells enabled="true" cell_size="2"/>
        <force_ped  a="5" D="0.2"/>
        <force_wall a="5" D="0.02"/>
    </model_parameters>
    <agent_parameters agent_parameter_id="1">
        <v0 mu="1.34" sigma="0.0" />
        <v0_upstairs mu="0.668" sigma="0.167" />
        <v0_downstairs mu="0.750" sigma="0.188" />
        <v0_idle_escalator_upstairs mu="0.5" sigma="0.0" />
        <v0_idle_escalator_downstairs mu="0.5" sigma="0.0" />
        <bmax mu="0.15" sigma="0.0" />
        <bmin mu="0.15" sigma="0.0" />
        <amin mu="0.15" sigma="0.0" />
        <atau mu="0." sigma="0.0" />
        <tau mu="0.5" sigma="0.0" />
```

```
        <T mu="1" sigma="0.0" />
    </agent_parameters>
</model>
```

Wall-avoidance model

Wall-avoidance model is a velocity-based model. The Wall-Avoidance Model focuses on valid pedestrian positions. The interaction of agents with walls takes precedence over the agent-to-agent interaction. There are two key aspects:

- In the vicinity to walls, agents take on a different behaviour, slowing them down (parameter: `slowdowndistance`)
- Agents follow special floorfields, directing them to the targets/goals, which will have them avoid walls if possible (free space)

Valid exit strategies are {6, 8, 9}. Please see details below.

(Sample) Usage:

```
<model operational_model_id="4" description="gradnav">
  <model_parameters>
    <solver>euler</solver>
    <stepsize>0.01</stepsize>
    <exit_crossing_strategy>9</exit_crossing_strategy>
    <floorfield delta_h="0.0625" wall_avoid_distance="0.4"
        use_wall_avoidance="true" />
    <linkedcells enabled="true" cell_size="4.2" />
    <force_ped nu="3" b="1.0" c="3.0" />
    <force_wall nu="1" b="0.70" c="3.0" />
    <anti_clipping slow_down_distance=".2" />
  </model_parameters>
  <agent_parameters agent_parameter_id="0">
    <v0 mu="1.5" sigma="0.0" />
    <bmax mu="0.25" sigma="0.001" />
    <bmin mu="0.20" sigma="0.001" />
    <amin mu="0.18" sigma="0.001" />
    <tau mu="0.5" sigma="0.001" />
    <atau mu="0.23" sigma="0.001" />
  </agent_parameters>
</model>
```

Parameters

- `<exit_crossing_strategy>[6, 8, 9]</exit_crossing_strategy>`
  The strategies 6, 8 and 9 differ only in the way the floorfield is created:

  - 6: one floorfield over all geometry (building); only in 2D geometries; directing every agent to the closest exit

  - 8: multiple floorfield-objects (one for every **room**); each object can create a floor field on the fly to a target line (or vector of lines) within the room; working in multi-floor-buildings; requires a router that provides intermediate targets in the same room

  - 9: (**recommended**) multiple floorfield-objects (one for every **subroom**); each object can create a floor field on the fly to a target line (or vector of lines)
    within the same subroom; working in multi-floor-buildings;
    requires a router that provides intermediate targets in the same subroom;

- `<floorfield delta_h="0.0625" wall_avoid_distance="0.4" use_wall_avoidance="tru`
  `/>`

  - The parameters define:

    * **delta_h**: discretization/stepsize of grid-points used by the floor field

    * **wall_avoid_distance**: below this wall-distance, the floor field will show a wall-repulsive character, directing agents away from the wall

    * **use_wall_avoidance**: {true, false} switch to turn on/off the enhancement of the floor field

- `<linkedcells enabled="true" cell_size="4.2" />`

  - range in which other pedestrians are considered neighbours and can influence the current agent. This value defines cell-size of the cell-grid.

Generalized Centrifugal Force Model with lateral swaying

The Generalized Centrifugal Force Model with lateral swaying is mostly identical to the GCFM Model,
but instead of a variable semi-axis b of the ellipse simulating the pedestrian, pedestrians perform an oscillation perpendicular to their direction of motion.
As a consequence the parameter `Bmax` is ignored.

Usage:

```
<model operational_model_id="5" description="krausz">
```

Four Parameters can be passed to control the lateral swaying, for example:

```
<sway ampA="-0.14" ampB="0.21" freqA="0.44" freqB="0.35" />
```

- `ampA` and `ampB` determine the amplitude of the oscillation according to the linear relation
  $A = \texttt{ampA} \cdot \|v_i\| + \texttt{ampB}$.

- `freqA` and `freqB` determine the frequency of the oscillation according to
  $f = \texttt{freqA} \cdot \|v_i\| + \texttt{freqB}$.

Setting `ampA` and `ampB` to 0 disables lateral swaying. If not specified, the empirical values given in Krausz, 2012 are used, that is:

- `ampA` = -0.14, `ampB` = 0.21 and

- `freqA` = 0.44, `freqB` = 0.25.

3.3 Direction strategies

The desired direction of a pedestrian is defined following different algorithms:
In the section of the chosen model the direction strategy should be specified as follows:

```
<exit_crossing_strategy>num</exit_crossing_strategy>
```

with *num* a positive integer.

The majority of the strategies define how a pedestrian crosses a line $L = [P_1, P_2]$. Possible values are:

1. The direction of the pedestrian is towards the middle of L ($\frac{P_1+P2}{2}$)

2. The direction is given by the nearest point on L to the position of the pedestrian.
   L is shorten by 20 cm.

3. If the nearest point of the pedestrian on the segment line L is outside the segment, then chose the middle point as target.
   Otherwise the nearest point is chosen.

4. This strategy is still beta. It assumes that the simulation scenario has no loops or U-shaped corridors.
   Pedestrians, target he exit, even if it is outside their visibility range. In case of intersection with walls or obstacles, the temporary direction is rotated away from the wall.

5. Does not exist.

6. -9. Strategies using floor fields (ff) (vector fields); one ff per target (door, line, ...)

- 6: This strategy does use a floor field rather than heading towards a point on a line segment.

  For more details see this talk [1] and the corresponding thesis [2].

[1] (https://fz-juelich.sciebo.de/index.php/s/s1ORGTUssCsHDHC)
[2] (https://fz-juelich.sciebo.de/index.php/s/VFnUCH2gtz1mSoL)

(**do not use in multi-storage buildings**)

- 7: (**experimental**)
- 8: This strategy uses a floor field collection for each room.

  Thus the floor fields are smaller but cannot steer to targets in a different room.

  The router **must** provide intermediate targets for every agent, the target being in the same room.

  The projection of the room onto the (x, y)-plane must be non-overlapping!
- 9: This strategy uses a floor field collection for each subroom. (**broken**)

  Thus the floor fields are again smaller but cannot steer to targets in a different subroom.

  The router **must** provide intermediate targets for every agent, that target being in the same subroom.

  The projection of the room onto the (x, y)-plane must be non-overlapping!

3.4 Route finding

Different router are implemented. However:

Floorfield Router

The floorfield-router is our latest router, which uses floorfields to calculate the distances among the doors of the same `subroom`. The major difference to any other router is, that it does **not** need convex subrooms/rooms any longer. There is no need for adding helplines.

It fills an adjacency matrix and calculates global-shortest paths via the Floyd-Warshall algorithm.

The floorfield-router will give intermediate targets within the`subroom` of each agent. It works in combination with exit strategies 8 and 9.[3]

If there are two points with the same (x, y)-coordinates, which differ only in the z-coordinate, the router will face problems, thus we defined the restriction above. That should avoid any such cases.

The floorfield router provides one mode: `ff_global_shortest`

`ff_local_shortest` and `ff_quickest` will follow shortly.

Following snippet is a definition example of the routing information:

```
<route_choice_models>
  <router router_id="1" description="ff_global_shortest">
  </router>

  <!-- Not yet implemented -->
  <!--router router_id="2" description="ff_local_shortest">
  </router-->
  <!-- Not yet implemented -->
  <!--router router_id="3" description="ff_quickest">
  </router-->
</route_choice_models>
```

[3] If convex subrooms are provided, any exit strategy will work. In these special cases, global router will be faster in computation time.

Global shortest path

At the beginning of the simulation, the Dijkstra algorithm is used to build a network which is then cached and used through the simulation life time.

Detailed information about the aforementioned models are presented in: KemlohWagoum2012a

Following snippet is a definition example of the routing information:

```
<route_choice_models>
  <router router_id="1" description="global_shortest">
    <parameters>
      <navigation_lines file="routing.xml" />
    </parameters>
  </router>
</route_choice_models>
```

The cognitive map

See this talk to get the idea

Smoke sensor documentation in JPSfire

```
<router router_id="7" description="cognitive_map">
  <sensors>
      <sensor sensor_id="1" description="Room2Corridor"/>
      <sensor sensor_id="2" description="Smoke" p_field_path="D:\
    JuPedSim\jpscore\inputfiles\cognitive_map\pFields\" update_time="
    30" final_time="300"/>
  </sensors>
  <cognitive_map status="complete" />
</router>
```

Updates

For current development updates, please check this issue on our GitLab Repo.

3.5 Validation and Verification

3.5.1 RiMEA tests

All these tests are described in more details in this article.
The notes give hints about specific implementation in JuPedSim.

- **Test 1**: One pedestrian is moving along a corridor.

  Test if pedestrian can maintain its speed constant

- **Test 2**: One pedestrian moving on a 10m long stair.

  It should be shown that the pedestrian can maintain its speed constant.

- **Test 3**: One pedestrian moving on a 10m long stair **downstairs**.

  It should be shown that the pedestrian can maintain its speed constant.

- **Test 4**: Show that the model can produce the shape of the fundamental diagram (density-velocity relation)
  in a simplified one-dimentional corridor with closed boundary conditions.

- **Test 5**: Distribute 10 pedestrians with 10 different reaction times.

  Verify whether they start exactly at the specified times.

- **Test 6**: 20 pedestrians going around a corner.

  Pedestrians should not cross walls.

- **Test 7**: Distribute pedestrian's speed according to the Table in Page 6. Four different groups are distributed according to their velocity:

  1. (v<30)

  2. (30<v<50)

3. (v>50)

4. handicapped.

Verify whether the speed values are within the specified range.

- **Test 8**: A 3D building is simulated and the influence of parameter e.g. speed is investigated.

  It should be shown how the evacuation time behaves with respect to the investigated parameter. [4]

- **Test 9**: 1000 pedestrians are distributed in a room with 4 exits.

  – scenario 1: All 4 exits are open

  – scenario 2: 2 exits are closed. The remaining 2 are still open

  The flow should nearly be doubled in scenario 1. [5]

- **Test 10**: Pedestrians are distributed in 12 different rooms. The building has two exits.

  The Pedestrians have exactly assigned exit numbers and should evacuate through these. [6]

- **Test 11**: 300 pedestrians are distributed in a room with two exits.

  The pedestrians should prefer the nearest exit, but some should (spontaneously) choose the second exit.

- **Test 12**: Two bottlenecks are connected with a long corridor.

  At the last exit there should be no jam. [7]

[4] This is not a fail criterion. It is just for documentation purposes.

[5] For simplicity in JuPedSim we simulate two identical rooms: - room left: with 4 exits. All of them are open - room right with 4 exits. two of them are closed - We write the trajectory in plain txt-format, to avoid a long lasting xml-parsing

[6] Pedestrian are assigned to two different groups. We verify if pedestrians in the two groups pass the exits. In the simulation pedestrians disappear once they are outside therefore we check if a pedestrian goes through line `exit - displacement`.

[7] The condition of this test is not clear enough... In the last exit there should be no jam, which means $J_{bottleneck} >= J_{last}$. However, this condition is not enough to quantify a jam.

- **Test 13**: Pedestrian coming out from a bottleneck along a corridor. At the end of the corridor is a stair.

  Since pedestrians have to reduce their speed on the stair, a jam should be observed at the beginning of the stair.

  In case of jam, flow at the beginning of the stair should be smaller than the flow in the corridor.[8]

- **Test 14**: Pedestrian's evacuation to an exit. Pedestrians have two possible routes:

  - short
  - and long

  This test has no concrete condition to check for.
  It should be documented whether pedestrians take a long detour or not
  There are 4 stats that should be documented:

  1. "kurz" (*short*)
  2. "lang" (*long*)
  3. "gemischt" (*mixed*)
  4. "konfigurierbar" (*configurable*)

Handbook	Speed Stair Up
PM	0.63 m/s
WM	0.61 m/s
NM	0.8 m/s
FM	0.55 m/s

Therefore, we choose for $v^0_{upstairs}$ a Gauss-distribution with $\mu = 0.675$ and $\sigma = 0.04$.

[8]The reduced speed on stairs (up) is according to Tab 1 in Burghardt2014.

3.5.2 Jülich tests

These tests are largely similar to the RiMEA tests.
They are extendend by **verification** tests to verify the validy of some algorithms implemented in JuPedSim and some **validation** tests based on experimental data.

Verification

- **Test 1**: : Free flow movement in a corridor

  A pedestrian that starts in the middle of a corridor (i.e. is not influenced by the walls)
  should move with its free flow velocity towards the exit.

- **Test 2**: One pedestrian moving in a corridor

  Rotating the same geometry as in test 1 around the $z-$axis by an arbitrary angle e.g. $45°$ should lead to the evacuation time of 10 s.

- **Test 3**: One pedestrian moving in a corridor with a desired direction

  A pedestrian is started from a random position in a holding area.
  This test should be repeated with different initial positions.

  Expected result: The pedestrians should be able to reach the marked goal in all repetitions of the test.

- **Test 4**: Single pedestrian moving in a corridor with an obstacle

  Two pedestrians are aligned in the same room. The second pedestrian from left is standing and will not move during the test.

  Expected result: Pedestrian left should be able to overtake the standing pedestrian

- **Test 5**: Single pedestrian moving in a very narrow corridor with an obstacle

This test is Similar to test 4. Two pedestrians are aligned in the same room. The second pedestrian from left is standing and will not move during the test. The corridor is narrow and does not allow passing of two pedestians without serious overlapping.

Expected result: Pedestrian left should stop without overlapping with the standing pedestrian.

- **Test 6**: single pedestrian moving in a corridor with more that one target

  A pedestrian is moving in a corridor with several intermediate goals.

  Expected result: The pedestrian should move through the different targets without a substantial change in its velocity i.e. with a desired speed of 1 m/s the distance of 10 m should be covered in 10 s.

- **Test 7**: route choice with different exits

  In this section the correct behavior of the implemented routing algorithms as well as the correct router-assignment are tested. The investigated geometry has three different exists.

  - Two different groups of pedestrians are randomly distributed in the inner room, such that the first group is nearer to E_1 than to E_2. The number of pedestrians in both groups is relatively small but is equal ($N_{\text{group 1}} = N_{\text{group 2}} = 10$). The router strategy is `local shortest`.

  - One group of pedestrians $N = 50$ is randomly distributed in the inner room. The router strategy is `local shortest`.

  - One group of pedestrians $N = 50$ is randomly distributed in the inner room. The router strategy is `global shortest`.

  Expected result: The pedestrians should be able to choose between the local shortest as well as the global shortest route.

- **Test 8**: visibility and obstacle
  The position of one pedestrian is initialized such that it has no direct view to the exit.

Expected result: The pedestrian should avoid the obstacle and exit the room without overlapping with the obstacle.

- **Test 9**: runtime optimization using parallelism

  Implementations that make use of parallel paradigms e.g. OpenMP are tested as follows:

  1. Distribute randomly 100 pedestrians in the gray area.
  2. Measure the flow through the bottleneck of the evacuation time i.e. the time necessary for the last pedestrian to leave the system.
  3. Repeat points 1 and 2 several times to get a certain statistical significance of the results.

  Expected results:

  - The simulation results (flow, evacuation time, ...) should be invariant with respect to the number of cores used.
  - The run time should scale with the number of cores.

- **Test 10**: runtime optimization using Verlet neighbor lists or cell-linked list method

  This test maybe specific for force-based models. In order to reduce the complexity of calculating interactions among N pedestrians from $\mathcal{O}(N^2)$, several techniques can be used to keep track of the neighbors.

  The edge length of one cell is bigger than the cut-off radius of the interaction force.

  - Expected results:
    The simulation results (flow, ...) should be the same, for the same initial conditions, with cell size bigger than the cut-off radius of the force-based model used.
    The simulation should fail for a cell size smaller than the cut-off radius of the forces.
    The simulation time should scale with the cell size.

- **Test 11**: Test the room/subroom construct [9]

  The same geometry is constructed differently
  The whole geometry is designed as a `rooms` (i.e. utility space)
  The geometry is designed by dividing the utility space in connected `subrooms`

  Distribute randomly pedestrians in all sub-rooms of the geometry and repeat the simulation to get a certain statistical significance.

  Expected results:
  The mean value of the evacuation times calculated from both cases should not differ.

- **Test 12**: Obstructed visibility

  Four pedestrians being simulated in a bottleneck. Pedestrians 0 and 1 have zero desired speed i.e. they will not move during the simulation whereas pedestrians 2 and 3 are heading towards the exit.

  The visibility between pedestrians 2 resp. 3 and 0 resp. 2 is obstructed by a wall resp. an obstacle.

  Expected results: Pedestrians 2 and 3 should not deviate from the horizontal dashed line.

- **Test 13**: Test if flow through bottleneck with respect to exit width is comparable to empirical values.

- **Test 14**: Uniform distribution of initial positions

  The initial distribution of the pedestrian should be uniform. In a square room ($100 \times 100\ m^2$) 2000 pedestrians are randomly distributed. The test is repeated 1000 times.
  Divide the room equidistantly in 10 regions with respect to x- and $y-$ axis and count the number of pedestrians in each square.
  This count should be roughly the same in all squares:

```
import numpy as np
import scipy, scipy.stats
import matplotlib.pyplot as plt

```

[9] This test should be removed after refactoring the geometry-class. There will be not `subroom` anymore.

```
filename = "./path/to/file.txt"
data = np.loadtxt(filename)

x = data[:,2]
y = data[:,3]

nx = plt.hist(x, bins=10)[0]
ny = plt.hist(y, bins=10)[0]

px = scipy.stats.chisquare(nx)[1]
py = scipy.stats.chisquare(ny)[1]
```

Expected result: The mean value of the 1000 p-values of the χ^2-test should be around 0.5.

- **Test 15**: Runtime of the code
  The purpose of this test is to make sure that the code is not getting slower! The test scenario is set up such that
  the **evacuation times is larger that the execution time**. This test is ideally run in form of a nightly build.

  Expected result: $t_{\text{evac}} \geq t_{\text{exec}}$.

Validation

- **Test 1**: 1D movement with periodical boundary

  The shape of the fundamental diagram (ρ, v) should be realistic (decreasing velocity with increasing density).

- **Test 2**: 2D unidirectional flow in corridor with periodical boundary

  The shape of the fundamental diagram (ρ, v) should be realistic (decreasing velocity with increasing density).

- **Test 3**: Unidirectional flow in corridor with open boundary

  The shape of the fundamental diagram (ρ, v) should be realistic (decreasing velocity with increasing density).

- **Test 4**: Unidirectional flow around a Corner

The fundamental diagram is measured in two different locations. Before the corner and after the corner.
The shape of the fundamental diagram (ρ, v) should be realistic (decreasing velocity with increasing density).

- **Test 5**: Flow through bottleneck

  The flow $J = N/\Delta t$, with N is the total number of participants in the run and Δt is the time interval between the first and the last pedestrian entering the bottleneck, should increase linearly with increasing width of the bottleneck.

- **Test 6**: Merging flow in T-junction

  The fundamental diagram is measured in three different locations.
  Right and left of the T-junction and after the merging of flows.

  The shape of the fundamental diagram (ρ, v) should be realistic (decreasing velocity with increasing density).

- **Test 7** - Bidirectional flow in corridor

  The shape of the fundamental diagram (ρ, v) should be realistic (decreasing velocity with increasing density).

4 JPSreport

4.1 File formats

`jpscore` needs as input an “inifile”, a geometry file and a trajectory file.

4.1.1 Inifile

To run `JPSreport` the only thing you need do is to copy a sample file and change the parameters in the file based your own situation.
Users can find template files in the folder “demos/” with the a name starting with “ini*”.

In the configuration file, the following five sections should be set:

- **Header**

```
<?xml version="1.0" encoding="UTF-8" ?>
<JPSreport project="JPS-Project" version="0.8" xmlns:xsi="http:
    //www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://xsd.jupedsim.org/0.6/
    jps_report.xsd">
```

- **geometry** indicates the file name corresponding to the trajectory files to analyze.
  Note that the file should be in the same location with the configuration file.

```
<geometry file = "geo_K0_240_050_240.xml" />
```

- **trajectories** indicates the location and the name of the trajectory files that will be analyzed.
  The format of trajectory files should be `.txt` or `.xml`.
  The supported unit of the trajectories is `m`.
  Two other sub-options `file` and `path` can be supplied.

If only `path` is given, then all files with the corresponding format in the given folder will be considered as the upcoming trajectories and `JPSreport` will try to load them one by one.
If both `file` and `path` are given, then only the given trajectories will be considered (several `file` tags can be given at the same time).

The location can be either absolute path or relative path to the location of the inifile.
A path is considered absolute if it starts with "/" (Linux system) or contains ":" (Windows system).

For example:

```
<trajectories format="txt" unit="m">
    <file name="traj_KO_240_050_240.txt" />
    <file name="traj_KO_240_060_240.txt" />
    <path location="./" />
</trajectories>
```

- **scripts** gives relative path based on the location of inifile or the absolute path.

```
<scripts location="../../scripts/"/>
```

- **measurement_areas** indicates the types and location of the measurement areas you
plan to use for analysis. Mainly Two kind of measurement areas can be exist here. `area_B` is 2D area and can be polygon (all the vertexes of the polygon should be given in clockwise), while `area_L` is a reference line indicating with two points. `area_L` is only used in 'method_A' and `area_B` are used for 'method_B, method_C and method_D'. Several measurement areas can be given and distiguished with different `id`.

  The parameter `zPos` is used to indicate the position of measuement area in z axis. `zPos` is useful for geometry with several stories.

  Notes:

  - the option `length_in_movement_direction` is only used in 'method_B' and the value will be ignored in other methods.

– If not given in ‘method_B’, the effective distance between entrance point to the measurement area and the exit point from the measurement area will be used.

```
<measurement_areas unit="m">
    <area_B id="1" type="BoundingBox" zPos="None">
        <vertex x="-2.40" y="1.00" /> <!-- Clockwise -->
        <vertex x="-2.40" y="3.00" />
        <vertex x="0" y="3.00" />
        <vertex x="0" y="1.00" />
        <length_in_movement_direction distance="2.0" />
    </area_B>
    <area_L id="2" type="Line" zPos="None">
        <start x="-2.40" y="1.00" />
        <end x="0" y="1.00" />
    </area_L>
    <area_L id="3" type="Line" zPos="None">
        <start x="-2.40" y="2.00" />
        <end x="0" y="2.00" />
    </area_L>
</measurement_areas>
```

- **velocity** precises the method for calculating the instantaneous velocity $v_i(t)$ of pedestrian i at time t from trajectories:

$$v_i(t) = \frac{X(t + \frac{frame\_step}{2}) - X(t - \frac{frame\_step}{2})}{frame\_step}.$$

```
<velocity frame_step="10" set_movement_direction="None"
    ignore_backward_movement="false"/>
```

Possible parameters are

– `frame_step` gives the size of time interval for calculating the velocity. The default value is 10.

– `set_movement_direction` indicates in which direction the velocity will be projected.
The value of `set_movement_direction` can be:

* `None`, which means that you don't consider the movement direction and
  calculate the velocity by the real distance. This is the default value.

* Any real number from `0` to `360` which represents the angular information
  of the direction in the coordination system.
  Note that the axis can be represented either by `X+`, `Y+`, `X-`, `Y-` or
  by 0, 90, 180, 270.

* `SeeTraj`. For complex trajectories with several times of direction change,
  you can indicate the detailed direction using the
  angular information in the trajectory file
  (By adding a new column in `.txt` file or adding a new variable in `.xml`
  file with the indicator `VD`).

– `ignore_backward_movement` indicates whether you want to ignore the movement
  opposite to the direction from `set_movement_direction`.
  The default value is `false`.

- **methods** indicates the parameters related to each measurement method.
  Four different methods `method_A` to `method_D` are integrated in the current version of `JPSreport` and can be chosen for the analysis.
  Further information relating to each method can be found
  in Pedestrian fundamental diagrams: Comparative analysis of experiments in different geometries.

  – Method A

```
<method_A enabled="true">
    <measurement_area id="2" frame_interval="100"
    plot_time_series="true"/>
    <measurement_area id="3" frame_interval="100"
    plot_time_series="true"/>
</method_A>
```

  Possible parameters are:

* `id` specifies the location of the reference line.
  several measurement areas can be set in one inifile with different id-numbers.

* `frame_interval` specifies the size of time interval (in *frame*) for calculating flow rate.

* `plot_time_series` specifies whether output the $(N - t)$-Diagram.

* Output data (in the folder: './Output/Fundamental_Diagram/FlowVelocity/'). See `Example-Method-A`:

– Method B

```
<method_B enabled="false">
    <measurement_area id="1" />
</method_B>
```

this method can only be used to analyze one directional (or part of one directional) pedestrian movement in corridors.
The speed is defined by the length of the measurement area `length_in_movement_direction` and the time a pedestrian stays in the area.
Possible parameters are:

* `measurement_area` given by an `id` number.
  Note that the measurement area for method_B should be rectangle based on the definition of the method.

* Output data: mean density and velocity of each pedestrian (ρ_i and v_i).

– Method C: Classical method

```
<method_C enabled="true">
    <measurement_area id="1" plot_time_series="false"/>
</method_C>
```

Possible parameters are:

* `id` indicates the size and location of the measurement_area.
  Several measurement areas can be set in one inifile.

* `plot_time_series` specifies whether output the $(\rho - t)$ and $(v - t)$ diagrams.

* Output data: mean density and velocity of over time $rho(t)$ and $v(t)$.

– Method D: Voronoi method

```
<method_D enabled="true">
    <measurement_area id="1" start_frame="None" stop_frame
="None" plot_time_series="false" get_individual_FD="false
"/>
    <one_dimensional enabled="false"/>
    <cut_by_circle enabled="false" radius="1.0" edges="10"
/>
    <output_voronoi_cells enabled="false" plot_graphs="
false"/>
    <profiles enabled="false" grid_size_x="0.20"
grid_size_y="0.20"/>
</method_D>
```

Possible parameters are:

* For each `measurement_area`, several id numbers can be set in one inifile.
  `start_frame` and `stop_frame` give the starting and ending frame for data analysis.
  The default values of these two parameters are `None`.
  If you plan to analysis the whole run from beginning to the end,
  set both of `start_frame` and `stop_frame` as `None`;
  If `start_frame =None` but `stop_frame` is not,
  then analysis will be performed from beginning of the trajectory to the `stop_frame`.
  If `start_frame` is not `None` but `stop_frame = None`,
  it will analyze from the `start_frame` to the end of the movement.
  `plot_time_series` specifies whether output the $\rho - t$ and $v - t$-diagram.
  `get_individual_FD` determines whether or not to output the data for individual fundamental diagram in the given measurement area, which is based on the Voronoi density ρ_i and velocity v_i of each pedestrian i in a given measurement area but not mean

value over space.
If true, the related data will be written in the
folder *./Output/Fundamental_Diagram/IndividualFD/*

* `one_dimensional` should be used when pedestrians move on a line
(for example, trajectories from
single-file experiment.

– `cut_by_circle` determines whether to cut each cell by circle or not.
Two options `radius` of the circle and the number of `edges` have
to be supplied for approximating the circle if `enabled` is *true*.

– `output_voronoi_cells` specifies whether or not to output data for visualizing
the Voronoi diagram.
Two options `enabled` and `plot_graphs` have to be set.
If both of them are `true`, files including Voronoi cells,
speed and the coordinates of pedestrian corresponding to each
cell as well as the figures of Voronoi cells will be created in
the folder *./Output/Fundamental_Diagram/Classical_Voronoi/VoronoiCell/*.
If the latter is `false`, only the data will be created but the figures
will not be plotted.
When `enable` is `false`, nothing will be created.

– `profiles` indicates whether to calculate the profiles over time and space.
If `enabled` is true, the resolution which is decided by the
parameters `grid_size_x` and `grid_size_x` should be set.
The data will be in the folder
./Output/Fundamental_ Diagram/Classical_Voronoi/field/.

– Output data: Mean density and velocity over time $\rho(t)$ and $v(t)$.
Sample data for plotting the Voronoi cells.
Data for plotting profiles.
Data of individual Fundamental diagram.

4.1.2 Trajectory file

JPSreport supports the formats .xml and .txt in current version. The format is the same with the output from JPScore.

Note the unit of the data in trajectory file should be all in m.

XML format

The file should include at least two main sections: header and frames.

- Header

```
<header version = "0.5">
   <agents> 1 </agents>
   <frameRate> 8 </frameRate>
</header>
```

  - agents: The total number of agents in the trajectory data.
  - frameRate: The frame rate.

- Frames: gives trajectory information in each frame.

```
<frame ID="0">
<agent ID="1" x="6.60" y="3.33" z="0.30"
              rA="0.17" rB="0.24" eO="-1.68" eC="0"/>
</frame>

<frame ID="1">
<agent ID="1" x="6.58" y="3.32" z="0.30"
              rA="0.31" rB="0.23" eO="-1.75" eC="54"/>
</frame>
```

  - **ID**: mandatory, the id of the pedestrians starting with 1.
  - **x, y, z**: mandatory, the position of the agent.

– **xVel, yVel, zVel**: Optional, the instantaneous velocity. They are not used in `JPSreport`.

– **rA, rB**: Optional. The shape which is defined by a circle (ellipse) drawn around a human. They are not used in `JPSreport`.

– **eO, eC**: Optional. They are the "ellipseOrientation" and the "ellipseColo"'. They are not used in `JPSreport`.

- Sample trajectory file

```
<?xml version="1.0" encoding="UTF-8"?>

    <trajectories>
        <header version = "0.5">
            <agents>1</agents>
            <frameRate>8</frameRate>
        </header>

        <frame ID="0">
            <agent ID="1" x="6.60"  y="3.33"  z="0.30"/>
        </frame>

        <frame ID="1">
            <agent ID="1" x="6.58"  y="3.32"  z="0.30"/>
        </frame>
    </trajectories>
```

TXT format

A sample trajectory in .txt format is present as below:

```
#description: optional description
#framerate: 16
#geometry: /home/sim/corridor.xml
#ID: the agent ID
#FR: the current frame
#X,Y,Z: the agents coordinates in meters

#ID FR  X   Y   Z
1 0 8.21  131.57  0.00
2 0 8.41  133.42  0.00
1 1 8.21  131.57  0.00
2 1 8.41  133.42  0.00
```

The line starting with `#framerate` and `#ID FR X Y Z` are necessary.

The order of each column is irrelevant. JPSreport will check the meaning of each column from the comments `#ID FR X Y Z`

Hint:
Use the script `scripts/txt2txt.py` to format a txt-trajectory according to the above-mentioned requirements.

4.2 Measurement methods

4.2.1 Method A

This method calculates the mean value of flow and density over time.

A reference line is taken and studied over a
fixed period of time Δt.
Using this method we can obtain the pedestrian flow J and the
velocity v_i of each pedestrian passing the reference line directly.
Thus, the flow over time $\langle J \rangle_{\Delta t}$ and the time mean
velocity $\langle v \rangle_{\Delta t}$
can be calculated as

$$\langle J \rangle_{\Delta t} = \frac{N^{\Delta t}}{t_N^{\Delta t} - t_1^{\Delta t}} \qquad \text{and} \qquad \langle v \rangle_{\Delta t} = \frac{1}{N^{\Delta t}} \sum_{i=1}^{N^{\Delta t}} v_i(t),$$

where $N^{\Delta t}$ is the number of persons passing the
reference line during the time interval Δt.

$t_N^{\Delta t}$ and $t_1^{\Delta t}$ are the times when the first and last pedestrians pass the location in Δt.

Note: this time period can be different from Δt.

The time mean velocity $\langle v \rangle_{\Delta t}$ is defined as the mean value of the instantaneous velocities $N^{\Delta t}$ pedestrians.

$v_i(t)$ is calculated by use of the displacement of pedestrian i in a small time interval $\Delta t'$ around t:

$$v_i(t) = \frac{\vec{x_i}(t + \Delta t'/2) - \vec{x_i}(t - \Delta t'/2))}{\Delta t'}.$$

4.2.2 Method B

This method measures the mean value of velocity and density over space and time. The spatial mean velocity and density are calculated by taking a segment Δx in a corridor as the measurement area.

The velocity $\langle v \rangle_i$ of each person
is defined as the length Δx of the measurement area divided by the time he or she needs to cross the area:

$$\langle v \rangle_i = \frac{\Delta x}{t_{\text{out}} - t_{\text{in}}},$$

where t_{in} and t_{out} are the times a person enters and exits the measurement area, respectively.

The density ρ_i for each person i is calculated as:

$$\langle \rho \rangle_i = \frac{1}{t_{\text{out}} - t_{\text{in}}} \cdot \int_{t_{\text{in}}}^{t_{\text{out}}} \frac{N'(t)}{b_{\text{cor}} \cdot \Delta x} dt,$$

where b_{cor} is the width of the measurement area while $N'(t)$ is the number of person in this area at a time t.

4.2.3 Method C

this method is the classical method for calculating the density.

The density $\langle \rho \rangle_{\Delta x}$ is defined as the number of pedestrians divided by the area of the measurement section:

$$\langle \rho \rangle_{\Delta x} = \frac{N}{b_{\text{cor}} \cdot \Delta x}.$$

The spatial mean velocity is the average of the instantaneous velocities $v_i(t)$ for all pedestrians in the measurement area at time t:

$$\langle v \rangle_{\Delta x} = \frac{1}{N} \sum_{i=1}^{N} v_i(t).$$

4.2.4 Method D

this method calculates the density based on Voronoi diagrams,
which are a special kind of decomposition of a metric space determined by distances to a specified discrete set of objects in the space.

At any time the positions of the pedestrians
can be represented as a set of points, from which the Voronoi
diagram can be generated.

The Voronoi cell area, A_i, for each person i can be obtained.

Then, the density and velocity distribution of the space ρ_{xy} and v_{xy} can be defined as

$$\rho_{xy} = 1/A_i \quad \text{and} \quad v_{xy} = v_i(t) \qquad \text{if}(x, y) \in A_i,$$

where $v_i(t)$ is the instantaneous velocity of each person.

The Voronoi density for the
measurement area is defined as:

$$\langle \rho \rangle_v = \frac{\iint \rho_{xy} dx dy}{b_{\text{cor}} \cdot \Delta x}.$$

Similarly, the Voronoi velocity is defined as:

$$\langle v \rangle_v = \frac{\iint v_{xy} dx dy}{b_{\text{cor}} \cdot \Delta x}.$$

5 Publications

5.1 Publications

`JuPedSim` is a software build for academic use only.
The models implemented within this framework were developed in different theses and articles that were published in journals or presented in conferences.

This is a list of some works that contributed to JuPedSim or used it to produce results.

Theses

- Rüping, Fabian
  "Computersimulation einer Schulevakuierung - Reproduzierbarkeit von Evakuierungszeiten und Fundamentaldiagrammen auf Treppen"
  Bachlor thesis, Wuppertal University, 2016
  Talk
  Thesis

- Axnich, Robert
  "Parameterstudie zur Evakuierung einer unterirdischen Personenverkehrsanlage mit dem Jülich Pedestrian Simulator"
  Master thesis, Wuppertal University, 2016
  Thesis:

- Hein, Ben
  "Sensitivity analysis of a perception-based route choice algorithm for a continuous evacuation model"
  Bachlor thesis, Cologne University, 2016
  Talk
  Thesis

- Osterkamp, Maximilian
  "Simulation der Raumung einer unterirdischen Personenverkehrsanlage mit

dem Julich Pedestrian Simulator"
Bachlor thesis, Wuppertal University, 2015
Thesis

- Graf, Arne
"Automated Routing in Pedestrian Dynamics"
Master thesis, Fachhochschule Aachen, 2015
Talk
Thesis

- Haensel, David
"A Knowledge-Based Routing Framework for Pedestrian Dynamics Simulation."
Diploma thesis, Technische Universitaet Dresden, 2014
Thesis

- Kemloh Wagoum, Armel Ulrich.
"Route Choice Modelling and Runtime Optimisation for Simulation of Building Evacuation,"
PhD Wuppertal University, 2013

- Zhang, Jun
Pedestrian fundamental diagrams: Comparative analysis of experiments in different geometries
PhD Wuppertal University, 2012

- Chraibi, Mohcine
Validated force-based modeling of pedestrian dynamics
PhD Cologne University, 2012

Peer-reviewed journal articles

1. Kemloh Wagoum, Armel Ulrich, Steffen, Bernhard, Seyfried, Armin, and Chraibi, Mohcine.
"Parallel Real Time Computation of Large Scale Pedestrian Evacuations."
Advances in Engineering Software 6061 (2013): 98103.

- Kemloh Wagoum, Armel Ulrich, Seyfried, Armin and Holl, Stefan.
"Modeling the Dynamic Route Choice of Pedestrians to Assess the Criticality of Building Evacuation."
Advances in Complex Systems 15, no. 3 (2012). doi:DOI: 10.1142/S0219525912500294.

- Kemloh Wagoum, Armel Ulrich, Chraibi, Mohcine, Mehlich, Jonas, Seyfried, Armin and Schadschneider, Andreas.
  "Efficient and Validated Simulation of Crowds for an Evacuation Assistant."
  Computer Animation and Virtual Worlds 23, no. 1 (2012): 315. doi:10.1002/cav.1420.

- Kemloh Wagoum, Armel Ulrich, Seyfried, Armin, Fiedrich, Frank, and Majer, Ralph.
  "Empirical Study and Modelling of Pedestrians' Route Choice in a Complex Facility."
  In Pedestrian and Evacuation Dynamics 2012, edited by Ulrich Weidmann, Uwe Kirsch, and Michael Schreckenberg, 25165.
  Springer International Publishing, 2014.

- Chraibi, Mohcine, Kemloh, Ulrich, Seyfried, Armin and Schadschneider, Andreas.
  "Force-Based Models of Pedestrian Dynamics."
  Networks and Heterogeneous Media 6, no. 3 (2011): 42542. doi:10.3934/nhm.2011.6.425.

- Chraibi, Mohcine, Seyfried, Armin and Schadschneider, Andreas.
  "Generalized Centrifugal Force Model for Pedestrian Dynamics."
  Physical Review E 82 (2010): 046111. doi:10.1103/PhysRevE.82.046111.

Conferences

- Chraibi, Mohcine and Zhang, Jun
  Tutorials about basic methods and tools on pedestrian dynamics, PED12, Hefei 2012

  - JuPedSim

  - JPSreport

- Lämmel, Gregor, Chraibi, Mohcine, Kemloh Wagoum, Armel Ulrich and Steffen, Bernhard
  Hybrid multi- and inter-modal transport simulation: A case study on large-scale evacuation planning
  In *Transport and Research Board (TRB)* 2015.

- Schröder, Benjamin, Haensel, David, Chraibi, Mohcine, Arnold, Lukas, Seyfried, Armin and Andresen, Erik
  Knowledge- and Perception-based Route Choice Modelling in Case of Fire

In **6th International Symposium on Human Behaviour in Fire 2015**
Human Behaviour in Fire Symposium 2015, Cambridge, United Kingdom, 28 Sep 2015 - 30 Sep 2015
327-338 (2015)

- Tordeux, Antoine, Chraibi, Mohcine and Seyfried, Armin
  Collision-free speed model for pedestrian dynamics
  In **Traffic and Granular Flow '15**, to appear.

- Andresen, Erik, Chraibi, Mohcine, Seyfried, Armin
  Wayfinding and cognitive maps for pedestrian models (Talk)
  In **Traffic and Granular Flow '15**, to appear.

- Kemloh Wagoum, Armel Ulrich, Chraibi, Mohcine, Zhang, Jun and Lämmel, Gregor.
  "JuPedSim: An Open Framework for Simulating and Analyzing the Dynamics of Pedestrians."
  In **3rd Conference of Transportation Research Group of India**, 2015.

- Boltes, Maik, Chraibi, Mohcine, Holl, Stefan, Kemloh Wagoum, Armel Ulrich, Lämmel, Gregor, Liao, Weichen, Mehner, Wolfgang, Tordeux, Antoine, and Zhang, Jun.
  "Experimentation, Data Collection, Modeling and Simulation of Pedestrian Dynamics."
  In **Statistics, Probability & Numerical Analysis 2014** - Methods & Applications, 2014.

- Kemloh Wagoum, Armel Ulrich, Steffen, Bernhard and Seyfried, Armin.
  "Runtime Optimisation Approaches for a Real-Time Evacuation Assistant."
  In **9th Conference on Parallel Processing and Applied Mathematics**, edited by R. Wyrzykowski, 7203:38695.
  Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2012.

- Chraibi, Mohcine, Freialdenhoven, Martina, Schadschneider, Andreas and Seyfried, Armin.
  Modeling the Desired Direction in a Force-Based Model for Pedestrian Dynamics,
  In **Traffic and Granular Flow '11**, edited by Kozlov E., Valery V., Buslaev, Alexander P., Bugaev, Alexander S., Yashina, Marina V., Schadschneider, Andreas and Schreckenberg, Michael
  Springer Berlin Heidelberg, July 5, 2012.

• Kemloh Wagoum, Armel Ulrich, and Seyfried, Armin.
"Optimizing the Evacuation Time of Pedestrians in a Graph-Based Navigation." In **Developments in Road Transportation**, edited by Mahabir Panda and Ujjal Chattararaj, 18896. Macmillian Publishers India Ltd, 2010.