Jülich Pedestrian Simulator

Forschungszentrum Jülich GmbH

User's Guide Version 0.7

Authors:

Ulrich Kemloh
Mohcine Chraibi
Jun Zhang

Stand: July 15, 2015

Contents

Listings

List of Figures

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.

Forschungszentrum Jülich GmbH makes no warranty, expressed or implied, to users of JuPedSim, and accepts no responsibility for its use. Users of JuPedSim assume sole responsibility for determining the appropriateness of its use; and for any actions taken or not taken as a result of analyses performed using this tool.

Users are warned that JuPedSim is intended for **academic** use only. This tool is an implementation of several computer models that may or may not have predictive capability when applied to a specific set of factual circumstances. Lack of accurate predictions by the models could lead to erroneous conclusions with regard of life safety.

Chapter 1

JuPedSim

The Jülich Pedestrian Simulator (JuPedSim) is a tool for simulating and analysing pedestrians motion. It consists at the moment of three modules which are loosely coupled and can be used independently. These are:

1. **JPScore**: the core module for computing the trajectories given a geometry and a start configuration.
2. **JPSvis**: the tool for visualising the trajectories
3. **JPSreport**: the tool for analysing the trajectories. Possible measurements include density, velocity, flow and profiles.

The description and use of each module is presented in the next chapters. The code is written in C++ and all modules can be compiled using *CMake* and a C++ compiler that supports the latest c++11 standard. For convenience binaries for Windows and Linux are provided.

Binaries:
The binaries are available from https://github.com/JuPedSim/JuPedSim/releases

Sources:
JuPedSim is developed at the Forschungszentrum Jülich in Germany and the bleeding edge code is hosted in their intern git repository at http://cst.version.fz-juelich.de. Visit the link and browse the public projects.

At the moment only specific tags (releases) are available on github at https://github.com/JuPedSim/JuPedSim. For more information, please contact info@jupedsim.org.

Showcase:

Some videos are available on our YouTube channel: https://youtube.com/user/JuPedSim

1.1 What's new in JuPedSim v0.7

JPSeditor (new module)

- New module for editing geometry files with import/export support for dxf files.

JPScore

- Framework validation using the RiMEA testcases (www.rimea.de). More tests are following.
- Risk tolerance factor (value $\in [0\ 1]$) for pedestrian. Pedestrians with high values are likely to take more risks.
- Sources for generating agents at runtime. Parameter are frequency (agents per seconds) and maximum number
- Option to color the pedestrians by group, spotlight, velocity, group, knowledge, router, final_goal, intermediate_goal.
- More control over the triangulation specially to avoid skinny triangles.
- Improved statistics. The flow curve for the different exits can be computed at runtime.
- Changelog file
- RiMeA validation test cases
- Unit tests are now based on the Boost testing engine

JPSreport

- Scripts for performing the plots are now included
- Added the option for specifying the location of scripts in configuration file
- Embedded python scripts (**_Plot_N_t.py**, **_Plot_timeseries_rho_v.py**) for plotting N-t diagram (Method A), time series of density/velocity diagram (Method C and D) and Voronoi diagrams (Method D)

- Added python script (**SteadyState.py**) for automatically detecting steady state of pedestrian flow based on time series of density and velocity. When plotting fundamental diagrams normally only data under steady state are used due to its generality
- Added python script (**_Plot_FD.py**) for plotting fundamental diagram based on the detected steady state
- Changed the data type of frame rate (fps) from integer to float
- Changed the way for dealing with pedestrian outside geometry. In old version JPSreport stops when some pedestrians are outside geometry but now it continue working by removing these pedestrians from the list
- More than one sub rooms in one geometry can be analysed independently

JPSvis

- new visualisation mode that display the structure of the geometry.

Previous release notes are listed in the Appendix C.

1.2 Documentation

Additional information, for instance build instructions with different IDEs and platforms are given at:

- https://cst.version.fz-juelich.de/jupedsim/jpsreport/wikis/home
- https://cst.version.fz-juelich.de/jupedsim/jpscore/wikis/home

Chapter 2

Running a simulation

Perform the following steps to run a simulation from scratch:

1. Download the latest version for your architecture from https://github.com/JuPedSim/JuPedSim/releases
2. Open one of the demos folder present in the unpacked archive. There are at least three files:
   - project file: usually ends with "_ini.xml"
   - geometry file: usually ends with "_geo.xml"
   - trajectory file: usually ends with "_traj.xml"
3. Drag and drop the trajectories file on JPSvis.exe (Windows) or JPSvis (Linux) if you want to visualize the pre-computed trajectories.
4. Drag and drop the project file on JPScore.exe (Windows) or JPScore (Linux) in the case you want to compute the trajectories first.
5. Make some modifications to the files and enjoy.

Note In order to prevent crashes that can sometimes hard to debug, make sure to validate all your input files against the supplied definition files

- *jps_geometry.xsd* for the geometry
- *jps_ini_core.xsd* for the simulation module
- *jps_events.xsd* for files containing the events
- *jps_report.xsd* for the reporting module
- *jps_routing.xsd* for the files containing additional navigation lines.

Chapter 3

JPScore

The core module for performing the simulation (i.e. computing the trajectories) is initialized with a project file containing all the necessary parameters for the simulation. All input files are XML-based. All distances are in metres (m), all times are in seconds (s) and all speeds consequently are in metres per seconds (ms^{-1}). In the following JPSCORE denotes the executable.

3.1 Project file (ini)

A program call looks like this:

```
> jpscore simulation_ini.xml
```

The same result is achieved by dragging and dropping the project file on the executable. The typical content of a project file (ini.xml) is described in Listing 3.1.

Listing 3.1: Structure of a project file

```
1
2  <?xml version="1.0" encoding="UTF-8" ?>
3
4  <JuPedSim project="JPS-Project" version="0.6"
5  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6  xsi:noNamespaceSchemaLocation="http://xsd.jupedsim.org/0.6/jps_ini_core.xsd">
7
8      <!-- header: seed , geometry, output format -->
9
10     <!-- files containing pre defined events e.g closing doors  -->
11     <events_file>   </events_files>
12
13     <!-- traffic information: e.g closed doors or smoked rooms -->
14     <traffic_constraints>   </traffic_constraints>
15
16     <!-- goals outside the geometry-->
```

```
17      <routing>   <goals/>    </routing>
18
19      <!--pedestrians information and distribution -->
20      <agents operational_model_id="2">
21          <agents_distribution></agents_distribution>
22      </agents>
23
24      <!-- operational models -->
25      <operational_models>
26          <model operational_model_id="1" description="gcfm"/>
27      </operational_models>
28
29      <route_choice_models>
30          <router router_id="1" description="global_shortest"/>
31      </route_choice_models>
32
33  </JuPedSim>
```

3.1.1 Header

The header comprises following elements:

- `<seed>12542</seed>`

  The seed used for the random number generator. If missing the number of seconds elapsed since Jan 1, 1970 (i.e. *time(NULL)*) is used.

- `<max_sim_time>200</max_sim_time>`

  The maximal simulation time in seconds. The default value is 900 seconds. The simulation will end after this time.

- `<num_threads>4</num_threads>`

  The number of threads to be used for the parallelisation by OpenMP. The maximal value is given by the number of cores on the target computer.

- `<events_file> events.xml </events_files>`

  A file containing different events that will be triggered during the simulation (see section 3.2).

- `<show_statistics>true</show_statistics>`

  With this option enabled, rooms egress times and door usages are shown at the end of the simulation. The flow values at exits are also written in a file (see chapter 5).

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

```
<!--<socket hostname="127.0.0.1" port="8989"/> -->
</trajectories>
```

format A detailed description of the formats are presented in the next section. The options are:

- **xml-plain**: the default output format in XML. It can lead to large files.
- **plain**: simple text format

fps defines the frame rate per second for the trajectories. Note that the step size for the simulation usually varies from 0.001 to 0.01 due to stability issues.

color_mode defines how the color of the agents should be calculated. The options are:

- **velocity**: the color is correlated to the velocity (This is the default mode).
- **group**: pedestrian in the same group have the same color.
- **spotlight**: all agents are grayed, except the agents with the spotlight attribute set to true.
- **knowledge**: the pedestrian having the same amount of knowledge have the same color (see section 3.2);
- **router**: the agents following the same route choice strategy have the same color.
- **final_goal**: the agents having the same final goal have the same color.
- **intermediate_goal**: the agents having the same intermediate goal have the same color.

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

Streaming Optionally to a file location a network address (ip/hostname and port) for streaming the results can also been provided.

- `<geometry>corner_geo.xml</geometry>`

  The location of the geometry file. All files locations are relative to the actual location of the project file.

3.1.2 Traffic constraints

This section defines the start constraints related to the traffic. At the moment doors can be open or close. This initial configuration can be altered during the simulation using events (see section 3.2).

Listing 3.2: Sample traffic constraints section

```
<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 door as defined in the geometry file (see section 3.4).

caption optional parameter defining the caption of the door.

state current state of the door. Possible values are **close** or **open**.

3.1.3 Goal definitions (routing)

Additional goals might be defined *outside* the geometry. They should not overlap with any walls or be located in a room. The best practice is to position them near to exits. Goals are defined with close polygons i.e. the last vertex is equal to the first one. A sample goal definition is presented in Listing 3.3.

Listing 3.3: Sample goals definition

```
<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>
```

3.1.4 Agents definition

This section defines different groups of pedestrians with individual characteristics. An example is given in Listing 3.4.

Listing 3.4: Sample agents parameter section

```
<agents operational_model_id="2">
    <agents_distribution>

        <group group_id="1" agent_parameter_id="1" room_id="0" subroom_id="0"
              number="10" />

        <group group_id="2" agent_parameter_id="2" room_id="0" subroom_id="1"
              number="20"     goal_id="1" router_id="2" patience="5"/>

        <group group_id="3" agent_parameter_id="1" room_id="0" number="200"
              goal_id="-1" router_id="2" patience="50" x_min="6.52" x_max="41"
              y_min="6.52" y_max="49"/>

        <group group_id="4" agent_parameter_id="1" room_id="0" number="1" start_x
              ="25.3" start_y="365.90"/>

        <group group_id="5" agent_parameter_id="1" room_id="0" subroom_id="1"
              number="30" pre_movement_mean="15" pre_movement_sigma="0.6"/>

    </agents_distribution>

   <agents_sources><!-- frequency in persons/seconds -->
       <source id="1" frequency="1" agents_max="50" group_id="0" caption="source
             1" />
       <source id="2" frequency="1" agents_max="5" group_id="2" caption="source 2
             " />
   </agents_sources>
</agents>
```

operational_model_id mandatory parameter defining the operational model to be used. Values are 1 (gcfm) or 2 (gompertz)

group_id mandatory parameter defining the unique id of the group.

agent_parameter_id mandatory parameter defining the agents characteristics. This is used to simulate children or elderly for instance. See subsection 3.1.6 for more information.

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

number mandatory parameter defining the number of agents to distribute.

subroom_id defines the id of the subroom (see section 3.4) where the agents should be distributed. If omitted then the agents are homogeneously distributed in the room. See the geometry section.

goal_id one of the *id* as defined in routing subsection 3.1.3. If omitted or "-1" then the shortest path to the outside is preferred by the agent.

router_id defines the route choice model to used. Those models are defined in the section subsection 3.1.7

patience in seconds influences the route choice behavior when using the quickest path router. It basically defines how long a pedestrian stays in jams before attempting a re-routing.

pre_movement_mean, pre_movement_sigma define the pre-movement time of the agents. Only after this time, the concerned agents will start moving.

start_x, start_y define the initial coordinate of the agents. This might be useful for testing. Note that these coordinates are ignored if *number* is different from 1.

x_min, x_max y_min, y_max define additional constraints where to distribute the agents. Not all variable must be defined. For instance [x_min="0", y_min="0"] will only distribute agents in the intersection of the first quadrant and the specified room/subroom.

3.1.5 Sources definition

A source of agents can also be specified in the agent section. An example is given in Listing 3.4. The following line will trigger the generation of 1 agent every second until 50 agents are generated. All the generation agents will inherit the properties of the group 0.

```
<source id="1" frequency="1" agents_max="50"
        group_id="0" caption="source 1" />
```

3.1.6 Operative models

JuPedSim comes with two implementations of force-based models:

- Generalized Centrifugal Force Model (GCFM), with the id=1
- Gompertz model, with the id=2

Each model definition has at least two sections:

- model_parameters. Here are the definition of the parameters specific to the model, for instance the force range.
- agent_parameters. Here are the definition of the parameter specific to the agents, for instance their size or the desired velocity, as far as accounted by the model. This allows to attribute different characteristics to different groups, for instance children or elderly.

GCFM: operational_model_id=1

For the definition of the GCFM the user is refereed to [3].

Listing 3.5: Sample definition of the GCFM

```
<operational_models>
    <model operational_model_id="1" description="gcfm">
        <model_parameters>
            <solver>euler</solver>
            <stepsize>0.01</stepsize>
            <exit_crossing_strategy>4</exit_crossing_strategy>
            <linkedcells enabled="true" cell_size="2.2" />
            <force_ped nu="0.3" dist_max="3" disteff_max="2" interpolation_width=
                "0.1" />
            <force_wall nu="0.2" dist_max="3" disteff_max="2" interpolation_width
                ="0.1" />
        </model_parameters>
        <agent_parameters agent_parameter_id="1">
            <v0 mu="0.5" sigma="0.0" />
```

```
            <v0_upstairs mu="1" sigma="0.0" />
            <v0_downstairs mu="0.5" 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="0.5" sigma="0.001" />
        </agent_parameters>
        <agent_parameters agent_parameter_id="2">
            <v0 mu="0.5" sigma="0.0" />
            <v0_upstairs mu="0.5" sigma="0.0" />
            <v0_downstairs mu="0.5" 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="0.5" sigma="0.001" />
        </agent_parameters>
    </model>
</operational_models>
```

operational_model_id unique identifier of this model. The id is referred in the agent definition (see subsection 3.1.4).

agent_parameter_id unique identifier for this group. The id is referred in the agent distribution definition (see subsection 3.1.4).

solver the Euler scheme is used to solve the differential equation resulting from the agents equation of motion.

stepsize integration step size in seconds. Be careful not to choose a large value otherwise the system might become unstable. The recommended value 10^{-3} s.

exit_crossing_strategy defines 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.
- **3**: Same as 2, only the line L is shortened on both edges by 20 cm.
- **4**: 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.

- **5**: This strategy is still beta. It assumes that the simulation scenario as no loops or U-shaped corridors. Pedestrians, are steered towards the exit, even if it is not within their visibility range.

Most of the aforementioned strategies are discussed in [2].

linkedcell The linked-cells reduce the runtime by optimizing the neighbourhood query of pedestrians. Chose *false* or a large *cell_size* if you want to consider all pedestrians as neighbours. This corresponds to a brute force approach summing over all other pedestrians, while calculating the repulsive forces. Depending on the number of pedestrians, the simulation can take some time [6].

v0 $\sim \mathcal{N}(\mu, \sigma)$.
The desired velocity on a plane v_0 is Gaussian distributed between $[\mu - \sigma, \mu + \sigma]$ with the parameters μ and σ. μ defaults to 1.24

v0_upstairs $\sim \mathcal{N}(\mu, \sigma)$.
The desired velocity when walking upstairs. The default value is v_0.

v0_downstairs $\sim \mathcal{N}(\mu, \sigma)$.
The desired velocity when walking downstairs. The default value is v_0.

escalator_upstairs $\sim \mathcal{N}(\mu, \sigma)$.
The speed of an escalator moving upstairs. The default value is v_0.

escalator_downstairs $\sim \mathcal{N}(\mu, \sigma)$.
The speed of an escalator moving downstairs. The default value is v_0.

v0_idle_escalator_upstairs $\sim \mathcal{N}(\mu, \sigma)$.
The speed of a pedestrian standing or moving upstairs on an idle escalator. The default value is v_0.

v0_idle_escalator_downstairs $\sim \mathcal{N}(\mu, \sigma)$.
The speed of a pedestrian standing or moving downstairs on an idle escalator. The default value is v_0.

bmax, bmin $\sim \mathcal{N}(\mu, \sigma)$.
The original published version of the model for the semi-axis b has two fundamental problems (See [3]):

- pedestrians cannot have a zero desired speed.
- the linear dependency of the speed is, in fact a simplification, but is not supported by empirical data.

Following modifications are implemented in JuPedSim. We consider the following relation $f(v) = a\exp(bv)$ for the speed v and fit the parameters a and b according to the empirical data of the 1D experiment [8] (See Figure 3.1).

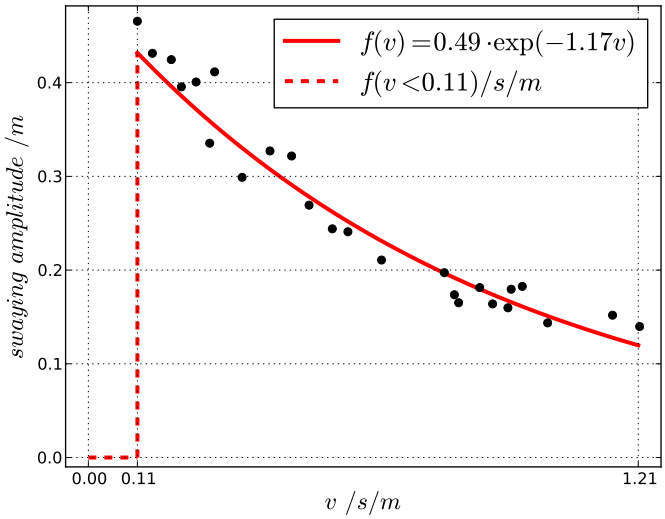


Figure 3.1: Fitting the amplitude of the swaying with respect to the speed of pedestrians.

Considering the semi-axis in the shoulder direction as the space required taken by one pedestrian while moving (including his lateral swaying) and assuming that the shoulder axis is $b_{\text{shoulder}} = 40$ cm, we set:

$$b(v) = \frac{1}{2}\Big(b_{\text{shoulder}} + f(v)\Big), \tag{3.1}$$

with $a = 0.49\,/m$, $b = -1.17/m/s$.

amin, atau $\sim \mathcal{N}(\mu, \sigma)$.
the length of the semi-axis in the direction of motion is defined by:

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

The parameter $a_{\min}$ and a_τ are Gaussian distributed.

force_ped, force_wall parameter for the repulsive force among pedestrians and with walls. See [3] for description of the parameter.

Gompertz model: operational_model_id=2

This model is unpublished but fully described in Appendix A. It is based on a continuous "physical" force. Many parameters defined for the GCFM are also valid for the Gompertz model. Both models differ only in the definition of the repulsive force (see Equation A.1 and Equation A.3).

Listing 3.6: Sample definition of the Gompertz model

```
<operational_models>
    <model operational_model_id="2" description="gompertz">
        <model_parameters>
            <solver>euler</solver>
            <stepsize>0.01</stepsize>
            <exit_crossing_strategy>3</exit_crossing_strategy>
            <linkedcells enabled="true" cell_size="2.2" />
            <bmax mu="0.25" sigma="0.01" />
            <bmin mu="0.20" sigma="0.01" />
            <amin mu="0.22" sigma="0.02" />
            <tau mu="0.5" sigma="0.00" />
            <atau mu="0.1" sigma="0.01" />
            <force_ped nu="3" b="0.25" c="3.0"/>
            <force_wall nu="10" b="0.70" c="3.0"/>
        </model_parameters>
        <agent_parameters agent_parameter_id="1">
            <v0 mu="1.3" sigma="0.0" />
            <v0_downstairs mu="1.0" sigma="0.0" />
            <v0_upstairs mu="0.5" 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="0.5" sigma="0.001" />
        </agent_parameters>
        <agent_parameters agent_parameter_id="2">
            <v0 mu="1.2" sigma="0.0" />
            <v0_upstairs mu="0.5" sigma="0.01" />
            <v0_downstairs mu="0.5" sigma="0.01" />
            <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="0.5" sigma="0.001" />
        </agent_parameters>
    </model>
</operational_models>
```

force_ped, force_wall parameter for the repulsive force among pedestrians and with walls. See Appendix A for a detailed description of the parameters.

- nu strength of the force
- b displacement along the x-axis $\equiv$ cut off radius
- c growth rate (y scaling).

3.1.7 Route choice models

Three route choice models are available at the moment. There are a lot more coming up with the next version. A sample definition is given in Listing 3.7.

Global shortest path

The default route choice algorithm is the global shortest path. At the beginning of the simulation, An evacuation network is generated based on the information present in the geometry file. This are mainly the location of the exits (and other navigation lines) and the states of the exits (closed/opened). The Dijkstra algorithm is then used to compute the shortest paths to the defined destinations. This algorithm is static and the agents do not adjust their routes, based for instance, on traffic conditions.

Quickest path

In this route choice algorithm, the agents are re-routed based on actual traffic conditions. The quickest path approach, which is based on the observation of the environment by the agents, is not sensitive to initial distribution of pedestrians or special topologies like symmetric exits, making it a general purpose algorithm.

The patience time can be used to control this behavior. This defaults to the global shortest path when the patience time of the agents are very large. Detailed information about the models are presented in [5].

Cognitive map

In this route choice algorithm, the agents are able to explore the geometry and collect various informations, for instance the location of the exits. For more information about the sensors and cognitive map see [4]. Concerning this router further specifications are necessary.

- It has to be specified whether the pedestrians start with an empty or complete cognitive map.
  - empty : The pedestrians don't know anything about the building or rather any route to the outside. They have to explore the building

room by room (depending on the used sensors they will make use of certain exit strategies i. e. look for corridors first).

  - complete : All rooms of the building are known. If no sensor is used pedestrians will follow the global shortest path (depending on spatial length).

- The user is able to decide which sensors are supposed to take into account

  - Sensor : Room2Corridor : Using this sensor the pedestrians will prefer to walk to (or stay on) a corridor.
  - Sensor : LastDestination : The pedestrians remember the destinations they already walk through. ‼Not using this sensor could lead to unrealistic behaviour if their cognitive maps are empty (pedestrian will stay at a crossing)‼
  - Sensor : Jam : A pedestrian will look for an alternative door (route) if a jam (under certain conditions) occurs in front of a door. ‼This sensor is not calibrated yet‼
  - The Sensor 'DiscoverDoors' is enabled by default as without it the pedestrian with an empty cognitive map couldn't find any exit route. Using a complete cognitive map this sensor won't be used.

- See Listing 3.7 to get information about how to specify cognitive map and sensors.

Listing 3.7: Sample route choice models

```
<route_choice_models>
    <router router_id="1" description="quickest">
        <parameters>
            <navigation_mesh method="triangulation"
                 use_for_local_planning="true" />
        </parameters>
    </router>

    <router router_id="2" description="global_shortest">
        <parameters>
             <navigation_lines file="routing.xml" />
        </parameters>
    </router>
    <router router_id="3" description="cognitive_map">

        <sensors>
            <sensor sensor_id="1" description="Room2Corridor" />
            <sensor sensor_id="2" description="Jam" />
            <sensor sensor_id="3" description="LastDestination" />
        </sensors>
        <cognitive_map status="empty"/>
```

```
22
23          <parameters>
24              <navigation_lines file="routing.xml" />
25          </parameters>
26      </router>
27  </route_choice_models>
```

Additional navigation lines In the case the geometry is not made of convex units (subrooms), additional navigations lines might be necessary to achieve a fluent motion for the agents, e.g at corners. An example file is given in Listing 3.8. The location of the file is given by:

```
<navigation_lines file="routing.xml" />
```

. Where routing.xml is the additional file containing navigation lines.

Navigation mesh (using triangulation) Alternatively a navigation mesh based on a delauney triangulation can be generated automatically.

```
<navigation_mesh method="triangulation"
 use_for_local_planning="true
 minimum_distance_between_edges="0.5"
 minimum_angle_in_triangles="20" />
```

With the value of $use\_for\_local\_planning$ set to $false$, the edges generated by the triangulation will only be used for computing distances and the pedestrians will be given exits only, and no intermediate destination. The navigation to that exit now depends on the $exit\_crossing\_strategy$ used (see subsection 3.1.6).

The options $minimum\_distance\_between\_edges$ and $minimum\_angle\_in\_triangles$ are used for filtering the generated triangles. since every vertex of the geometry is used in the triangulation, it might results to a large amount of skinny triangles. By using these options, distances and angles are used to filter out some of the edges.

Listing 3.8: Sample navigation lines file

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <routing version="0.5">
3      <Hlines>
4          <Hline id="2" room_id="0" subroom_id="1">
5              <vertex px="15.0" py="4.0" />
6              <vertex px="17.0" py="6.0" />
7          </Hline>
8          <Hline id="3" room_id="0" subroom_id="1">
9              <vertex px="15.0" py="4.0" />
```

```
            <vertex px="15.0" py="6.0" />
        </Hline>
        <Hline id="4" room_id="0" subroom_id="1">
            <vertex px="15.0" py="4.0" />
            <vertex px="17.0" py="4.0" />
        </Hline>
    </Hlines>
</routing>
```

- **id** mandatory unique identifier for this navigation line.
- **room_id** the room containing the navigation line.
- **subroom_id** the subroom containing the navigation line.
- **vertex** the coordinates of the navigation line.

3.2 Events handling

The simulation can be steered from outside using predefined events. At the moment, doors can be closed or opened. A sample file is given in Listing 3.9.

Listing 3.9: Sample events file

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

<JPScore project="with-events" version="0.6"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://xsd.jupedsim.org/0.6/jps_events.xsd">
  <events update_time="2" update_radius="1"
    agents_color_by_knowledge="true">
    <event time="15"   id="8" type="door" state="close" caption="left_exit" />
    <event time="50"   id="8" type="door" state="open" caption="left_exit" />
    <event time="50"   id="4" type="door" state="close" caption="main_exit" />
    <event time="100"  id="4" type="door" state="open" caption="main_exit" />
  </events>
</JPScore>
```

- **time** defines when the event should occur in seconds.
- **id** the identifier of the object to perform the action on.
- **type** the type of the object. At the moment only **door** is supported.
- **state** the next state to change in. Options are **close** and **open**

- **caption** optional caption of the object.
- **update_radius and** and **update_time** define the information propagation speed between the agents.
- **agents_color_by_knowledge** Useful for the visualisation. The agent having the same information have the same color. Other options are: agents_color_by_velocity (which is the default one) and agent_color_by_router.

Agents notice the state of a door when in the range (actually $0.5m$) of that door. That knowledge is then shared with other agents within update_radius. The process is repeated at a period of update_time leading to an information propagation speed of

$$v = \frac{update\_radius}{update\_time}\, ms^{-1}$$

The knowledge synchronization between two agents is based on the time stamps. If both agents have the same information, then the most recent one is kept. A transfer probability of 1 is assumed, meaning that the newest information is always accepted.

3.3 Sample project file

A complete example of a project file for the simulation is given in Listing 3.10. More examples are found in the demos folder which comes along side with the sources.

Listing 3.10: Sample project file

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

<JuPedSim project="JPS-Project" version="0.6"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://xsd.jupedsim.org/0.6/jps_ini_core.xsd">

    <!-- seed used for initialising random generator -->
    <seed>12542</seed>
    <!-- geometry file -->
    <geometry>bottleneck_geo.xml</geometry>
    <!-- traectories file and format -->
    <trajectories format="xml-plain"  fps="8">
        <file location="bottleneck_traj.xml" />
    </trajectories>
    <!-- where to store the logs -->
    <logfile>log.txt</logfile>

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

```
        <!-- doors states are: close or open -->
        <doors>
            <door trans_id="2" caption="main_exit" state="open" />
        </doors>
    </traffic_constraints>

    <routing>
        <goals>
            <goal id="0" caption="goal 1">
                <polygon>
                    <vertex px="70" py="101" />
                    <vertex px="70" py="103" />
                    <vertex px="75" py="103" />
                    <vertex px="75" py="101" />
                    <vertex px="70" py="101" />
                </polygon>
            </goal>
        </goals>
    </routing>

    <!--persons information and distribution -->
    <agents operational_model_id="1">
        <agents_distribution>
            <group group_id="1" agent_parameter_id="1" room_id="1" subroom_id="0"
                   number="40" goal_id="0" router_id="1" />
        </agents_distribution>
    </agents>

     <!-- These parameters may be overwritten -->
     <operational_models>
        <model operational_model_id="1" description="gcfm">
          <model_parameters>
            <solver>euler</solver>
            <stepsize>0.01</stepsize>
            <exit_crossing_strategy>4</exit_crossing_strategy>
            <linkedcells enabled="true" cell_size="2.2" />
            <force_ped nu="0.3" dist_max="3" disteff_max="2" interpolation_width=
                  "0.1" />
            <force_wall nu="0.2" dist_max="3" disteff_max="2" interpolation_width
                  ="0.1" />
          </model_parameters>
          <agent_parameters agent_parameter_id="1">
            <v0 mu="0.5" 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="0.5" sigma="0.001" />
          </agent_parameters>
          <agent_parameters agent_parameter_id="2">
            <v0 mu="0.5" 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="0.5" sigma="0.001" />
          </agent_parameters>
        </model>

        <model operational_model_id="2" description="gompertz">
          <model_parameters>
                <solver>euler</solver>
```

```
79                  <stepsize>0.01</stepsize>
80                  <exit_crossing_strategy>3</exit_crossing_strategy>
81                  <linkedcells enabled="true" cell_size="2.2" />
82                  <force_ped nu="3"  b="0.25" c="3.0"/>
83                  <force_wall nu="10" b="0.70" c="3.0"/>
84          </model_parameters>
85          <agent_parameters agent_parameter_id="1">
86                  <v0 mu="0.5" sigma="0.0" />
87                  <bmax mu="0.25" sigma="0.001" />
88                  <bmin mu="0.20" sigma="0.001" />
89                  <amin mu="0.18" sigma="0.001" />
90                  <tau mu="0.5" sigma="0.001" />
91                  <atau mu="0.5" sigma="0.001" />
92          </agent_parameters>
93          <agent_parameters agent_parameter_id="2">
94                  <v0 mu="0" sigma="0.0" />
95                  <bmax mu="0.25" sigma="0.001" />
96                  <bmin mu="0.20" sigma="0.001" />
97                  <amin mu="0.18" sigma="0.001" />
98                  <tau mu="0.5" sigma="0.001" />
99                  <atau mu="0.5" sigma="0.001" />
100         </agent_parameters>
101     </model>
102 </operational_models>
103
104 <route_choice_models>
105     <router router_id="1" description="global_shortest">
106         <parameters>
107      <navigation_lines file="routing.xml" />
108         </parameters>
109     </router>
110 </route_choice_models>
111
112 </JuPedSim>
```

3.4 Geometry

The general structure of the geometry file is shown in Figure 3.2. A sample geometry code is given in Listing 3.11 and the corresponding visualisation in Figure 3.3.

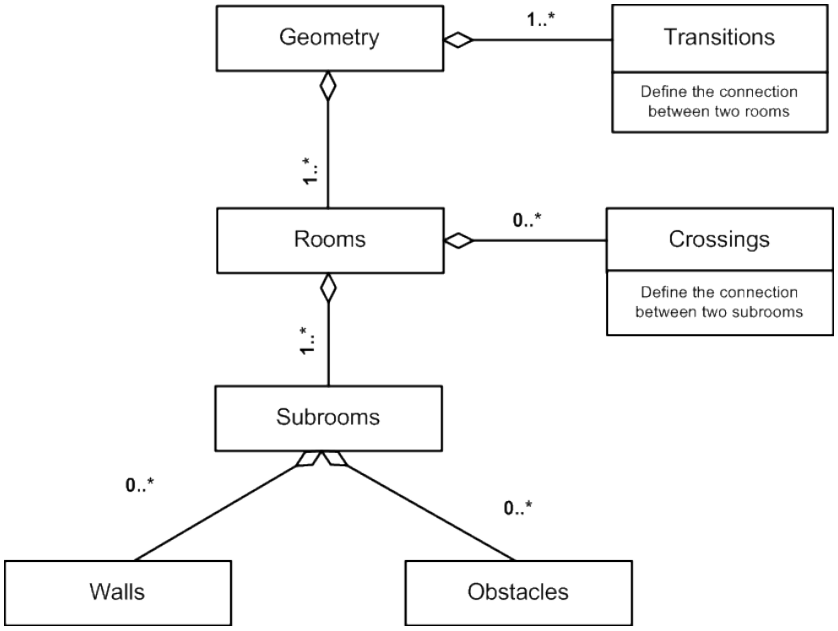


Figure 3.2: Structure of the geometry file.

Listing 3.11: Sample geometry file with one obstacle

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<geometry version ="0.5" caption="corner"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://xsd.jupedsim.org/jps_geometry.xsd">
    <rooms>
        <room id="0" caption="hall" >
            <subroom id="0"  class="subroom">
                <polygon caption="wall">
                    <vertex px="10.0" py="4.0"/>
                    <vertex px="10.0" py="0.0"/>
                    <vertex px="0.0" py="0.0"/>
                    <vertex px="0.0" py="10.0"/>
                    <vertex px="10.0" py="10.0"/>
                    <vertex px="10.0" py="6.0"/>
                </polygon>
            </subroom>
            <subroom id="1"  class="corridor">
                <polygon caption="wall">
                    <vertex px="10.0" py="6.0"/>
                    <vertex px="17.0" py="6.0"/>
                    <vertex px="17.0" py="-5.0"/>
                </polygon>
                <polygon caption="wall">
                    <vertex px="15.0" py="-5.0"/>
                    <vertex px="15.0" py="4.0"/>
                    <vertex px="10.0" py="4.0"/>
                </polygon>

                <obstacle id="0" caption="table" height="1.0" >
```

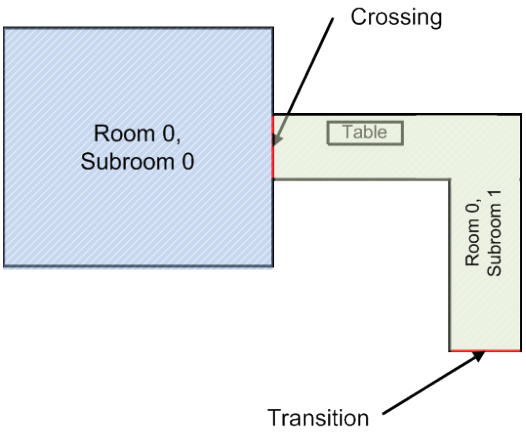


Figure 3.3: Sample geometry with one room, two subrooms and one obstacle. The obstacle should not overlap with any wall of the geometry.

```
30                      <polygon>
31                          <vertex px="12.0" py="6.0"/>
32                          <vertex px="13.0" py="6.0"/>
33                          <vertex px="13.0" py="5.5"/>
34                          <vertex px="12.0" py="5.5"/>
35                          <vertex px="12.0" py="6.0"/>
36                      </polygon>
37                  </obstacle>
38              </subroom>
39              <crossings>
40                  <!-- virtual exits between subrooms -->
41                  <crossing id="0" subroom1_id="0" subroom2_id="1">
42                      <vertex px="10.0" py="6.0"/>
43                      <vertex px="10.0" py="4.0"/>
44                  </crossing>
45              </crossings>
46          </room>
47      </rooms>
48
49      <transitions>
50          <!-- exits like crossings but between rooms or to outside (room with
                index
51              = -1) -->
52          <transition id="1" caption="main exit" type="emergency"
53              room1_id="0" subroom1_id="1" room2_id="-1" subroom2_id="-1">
54              <vertex px="15.0" py="-5.0"/>
55              <vertex px="17.0" py="-5.0"/>
56          </transition>
57      </transitions>
58  </geometry>
```

3.4.1 Rooms

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

3.4.2 Subrooms

The subrooms define the navigation mesh, i.e the walkable areas in the geometry. Each subroom is bounded by at least 1 crossing. A sample subroom definition is described in Listing 3.12. 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, see Listing 3.8.

- **id** mandatory parameter, also referred by crossings.
- **class** optional parameter defining the class of this subroom. At the moment two classes are defined:
  - **floor**
  - **stairs** takes an additionally

    ```
    <up px="-5.0" py="2" />
    <down px="0.0" py="2" />
    ```

    used in the visualisation, for marking the highest and the lowest point.
- **A_x, B_y, C** optional parameter for the explicit plane equation of the subroom. It allows the construction of a 3D environment (See Appendix B) 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: P1(1,0,0), P2 (0,1,0) and P3(0,0,1) then the equation is given by:

  $$Z = -x - y + 1$$

  .
- **polygon** describes the walls as a sequence of vertex.

Listing 3.12: Sample stair subroom

```
<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>
```

3.4.3 Transitions

A *transition* defines the connection between two rooms and is basically a door. They can be close or open in the configuration in the project file. See subsection 3.1.2.

Listing 3.13: Sample 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 for this door. The id is also used to close or open the door in the traffic constraints section of the project file. See subsection 3.1.2.
- **caption**, optional and displayed in the visualisation.
- **type**, optional. The parameter is not internally used.
- **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**, the geometry of the transition as a straight line.

3.4.4 Crossings

A *crossing* defines the connection between two subrooms inside the same room. Unlike transition, they cannot be close or open. They are always open.

Listing 3.14: 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. The id is also used to close or open the door in the traffic constraints section of the project file. See subsection 3.1.2.
- **subroom1_id**, the first subroom located in room_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**, the geometry of the transition as a straight line.

3.4.5 Obstacles

One or more obstacles can also be defined within a subroom. They should completely reside inside the subroom. Especially there should not any intersections with other geometry elements (walls, crossings or transitions).

Listing 3.15: 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.

3.5 Trajectories

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, the format is forced to xml-plain.

3.5.1 xml-plain

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

Header

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

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

Geometry

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>
```

Frames

```
<frame ID="0">
<agent ID="1"   x="660.00"   y="333.00"   z="30.00"
    rA="17.94"   rB="24.94"   eO="-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"   eO="-175.41" eC="54"/>
</frame>
```

- **ID** mandatory, the id of the pedestrians starting with 1.
- **x, y, z** mandatory, 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 modelled pedestrians’ shape is an ellipse. Else if it is a circle those values should be equal to the radius of the circle.
- **eO, 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 rapport between the desired velocity (V_0) and the instantaneous velocity.

Sample trajectory file

A sample trajectory in the xml format is presented in Listing 3.16.

Listing 3.16: Sample trajectory file in XML format

```
<?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"  eO="-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"  eO="-175.41"  eC="54"/>
    </frame>
</trajectories>
```

3.5.2 plain

A sample trajectory in the plain format is presented in Listing 3.17. The second line is always the *framerate*.

Listing 3.17: Sample trajectories file in plain text format

```
#description: my super 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
```

```
24                  2    7    38.85    133.42    0.00
25                  1    8    28.77    131.57    0.00
```

Chapter 4

Verification and validation

In order to control the quality of this software we are setting up a list of verification and validation tests. The list of the tests is continuously growing. The verification tests are based on tests published in the literature [9, 1, 10]. The goal of the verification tests is to ensure a minimal functionality of the software, whereas the validation tests make an assessment of the "realism" of the underlying models (relatively to experimental findings).

4.1 Set up a test

Several validation and verification tests for JuPedSim are defined in the following section. In order to make the nightly builds run automatically, consider the following steps, before adding new tests. This procedure is also recommended to make simulations with several inifiles e.g. different seeds.

For every test create a directory in *Utest/*

1. Find out the number of the last existing test. Let's assume that would be test_100.
2. Copy the directory test_1 into a directory, e.g. test_101
3. Rename the file runtest1.py into runtest_101.py (101 being the number of the test as in point 1.)
4. Adapt the new script, especially by changing the old *testnumber* 1 to the new test number (in this example 101) writing some smart tests.
5. *Git add* your directory test_101 after defining your test. The test will automatically be considered for the next nightly buid by CTest.

Note: the geometry file should always be called ”geometry.xml”. The geometry location in the master_ini.xml file is then

Listing 4.1: Structure of a project file

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

(the “..” is not a typo)

In the master inifile you can use python syntax to give different values for the following tags:

- **max_sim_time**: Max time of simulation
- **seed**: You should always change this one to get credible results
- **geometry**: Give a directory name containing different xml-files. In case you have only one file, just specify its name.
- **num_threads**: (Optional) Number of CPUs to be used in the simulation. If not given (and openmp is activated), the maximum of cores in the machine is used. Otherwise num_threads=1.
- **exit_crossing_strategy**: Strategy for the desired direction
- **stepsize**: Integration step size.

The following attributes can also be duplicated

- **operational_model_id**: operational model
- **number**: Number of agents
- **cell_size**: Cell size
- **pre_movement_mean**: mean value of the premovement time
- **pre_movement_sigma**: sigma value of the premovement time
- **agent_parameter_id**:
- **mu, sigma**: mu and sigma for all attributes that are Gauss-distributed, e.g.$v0$, $bmax$, $\cdots$
- **force_ped**: all parameters of the ped-ped repulsive forces.

- **force_wall**: all parameters of the ped-wall repulsive forces.

Example:

```
    <num_threads>[3,4]</num_threads>
    <seed>range(1, 10)</seed>
     <v0_upstairs mu="[0.5, 0.6, 0.7]" sigma="0.0" />
```

run the script makeini.py with the obligatory option **-f**: Using the aforementioned example the call is:

```
python makeini.py -f test_case1/inifile.xml
```

The script creates two directories:

- **test_case/trajectories** will be the location of the trajectories
- **test_case/inifiles** will be the location of the project files, that will be produced based on the master inifile (in this case test_case1/inifile.xml). Note, that the geometry file and the trajectory files are all relative to the project files in the directory *inifiles*.

Note that in the test scripts (runtest_n, $n \in \mathbb{N}$) the $makeini.py$ is called automatically.

4.2 Verification

This section describes the tests that should be passed by the simulation framework in order to guarantee a minimal usability. The tests are automated with CTest (www.cmake.org). We are continuously adding more tests.

4.2.1 Test 1: One pedestrian moving in a corridor

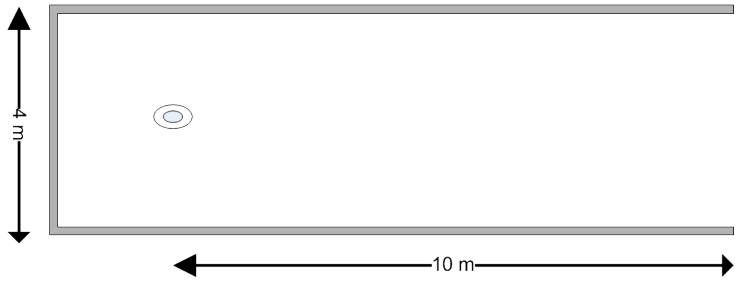


Figure 4.1: Test 1 horizontal corridor

Expected result: Pedestrian should exit the corridor within 10 seconds, if the desired speed is set to 1 m/s.

4.2.2 Test 2: One pedestrian moving in a corridor

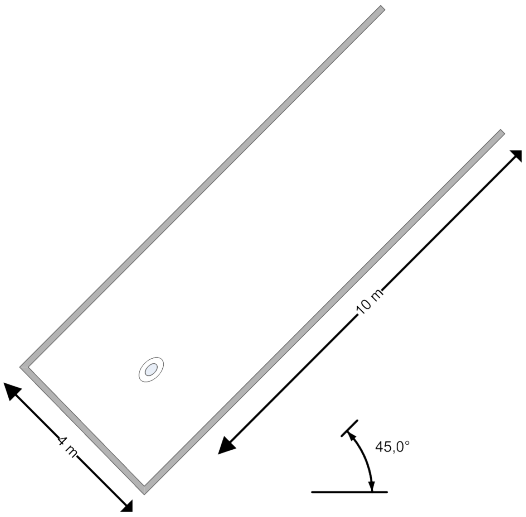


Figure 4.2: Test 2 inclined corridor 45 (2D)

Expected result: Pedestrian should exit the corridor within 10 seconds, if the desired speed is set to 1 m/s.

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

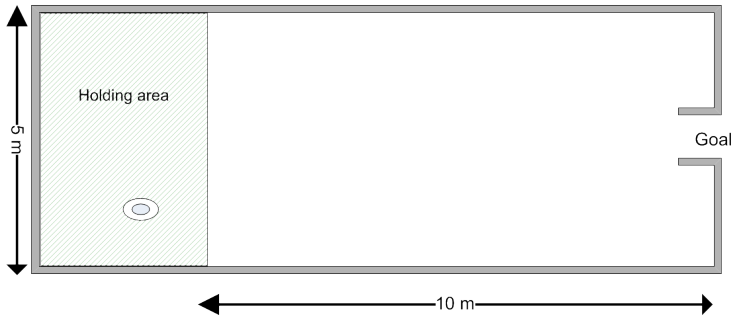


Figure 4.3: Test 3 pedestrian moving in a corridor with a desired direction

Expected result: from any random starting position in the holding area, the pedestrians should reach the marked goal.

4.2.4 Test 4: single pedestrian moving in a corridor with an obstacle

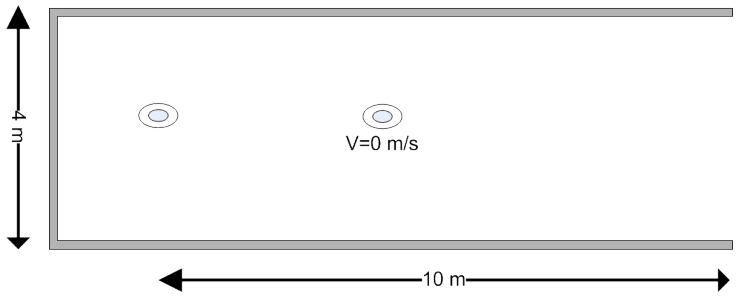


Figure 4.4: Test 4

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

4.2.5 Test 5: single pedestrian moving in a very narrow corridor with an obstacle

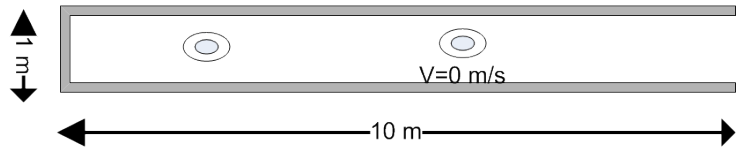


Figure 4.5: Test 5 single pedestrian moving in a very narrow corridor with an obstacle

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

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

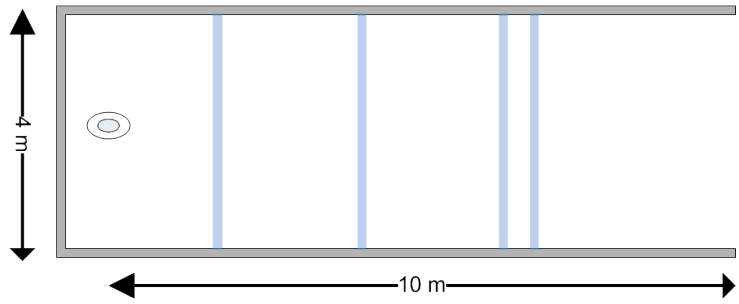


Figure 4.6: Test 6 pedestrian moving in a corridor with more that one target

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

4.2.7 Test 7: route choice with different exits location

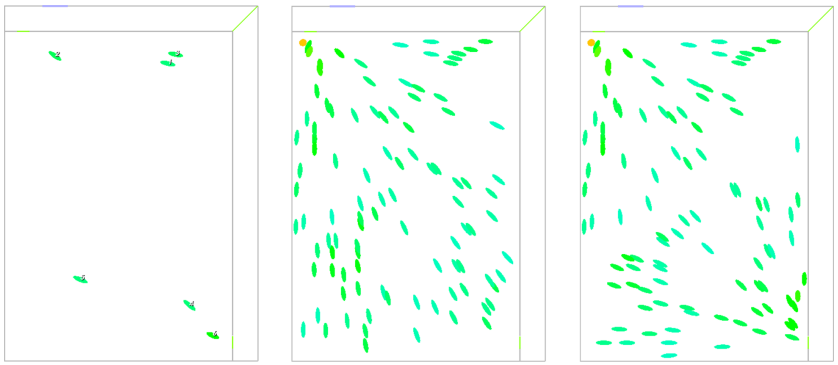


Figure 4.7: Test 7: route choice with two exits. (a): 2 groups of pedestrians head to two different exists (local shortest). (b): One group of pedestrians heads to one exist (global shortest). (c): One group of pedestrians heads to two different exists (local shortest).

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

4.2.8 Test 8: visibility and obstacle

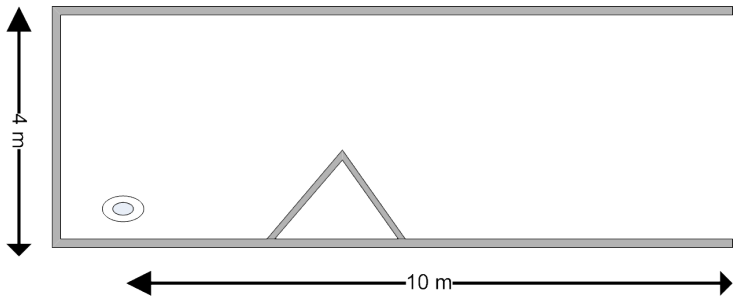


Figure 4.8: Test 8 visibility and obstacle

Expected result: The pedestrian should avoid the obstacle.

4.2.9 Test 9: runtime optimization using OpenMP

Expected results:

1. The simulation results (flow,...) should be the same for the same initial conditions independent on the number of cores used.
2. The run time should scale with the number of cores.

4.2.10 Test 10: runtime optimization using neighborhood list

Expected results:

1. 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.
2. The simulation should fail for a cell size smaller than the cut-off radius of the forces.
3. The simulation time should scale with the cell size.

4.2.11 Test 11: Test the room/subroom construct

The same geometry is constructed differently:

- using only the concept of “rooms”
- using the concept of “subrooms”

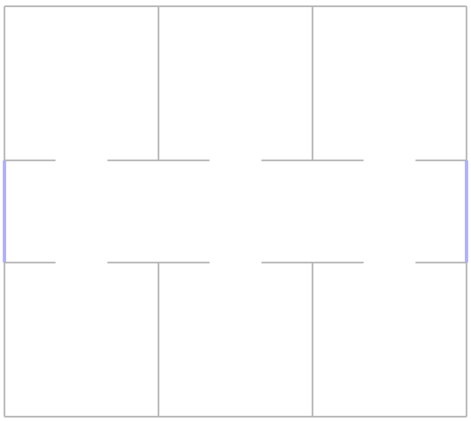


Figure 4.9: Test 11: This geometry can be constructed in two ways.

Expected results: The evacuation times calculated from both cases should be the same.

4.2.12 Test 12: Test visibility

Four pedestrians being simulated in a bottleneck. See Fig. 4.10. Pedestrians 1 and 3 have zero desired speed i.e. they so not move whereas pedestrians 2 and 4 are heading towards the exit.

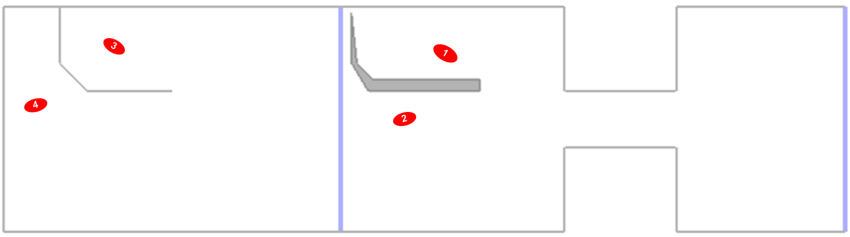


Figure 4.10: Test 12: The visibility of some pedestrians is strongly reduced by walls and/or obstacles.

Expected results: Pedestrians 2 and 4 should not be influenced by pedestrians 1 and 3 since they are out if sight.

4.2.13 Test 13: Flow through bottleneck

Expected result: The flow should increase linearly with respect to the width of the bottleneck.

4.2.14 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. See fig. 4.11

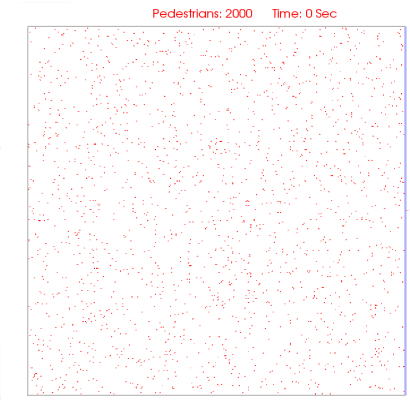


Figure 4.11: Test 14: 2000 pedestrians randomly distributed in a rooms.

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.

Expected result: The mean value of the 1000 p-values of the χ^2-test should be around 0.5. See Fig. 4.12.

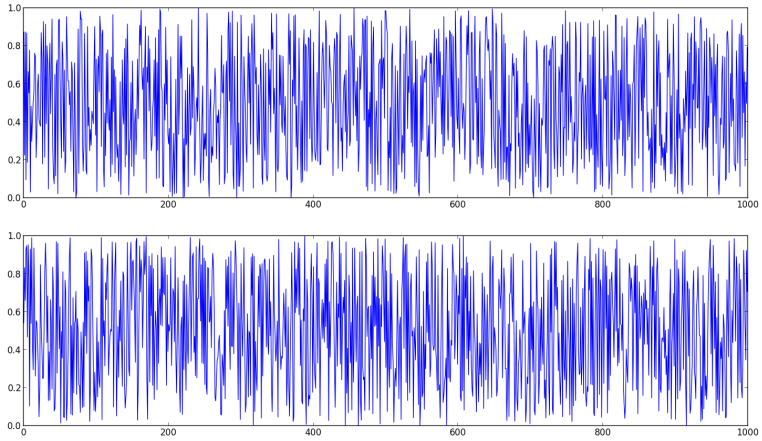


Figure 4.12: Test 14: p-values with respect to x and y.

4.2.15 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 time** is larger that the **execution time**.

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

4.3 Validation

This section describes the tests cases that are compared with empirical data. It is still under development.

Chapter 5

JPSreport

This module focuses on the analysis of pedestrian characteristics using trajectories obtained from video recordings as well as simulations. JPSreport integrates four different measurement methods described in [?]. The typical workflow using JPSreport is shown in Figure 5.1.

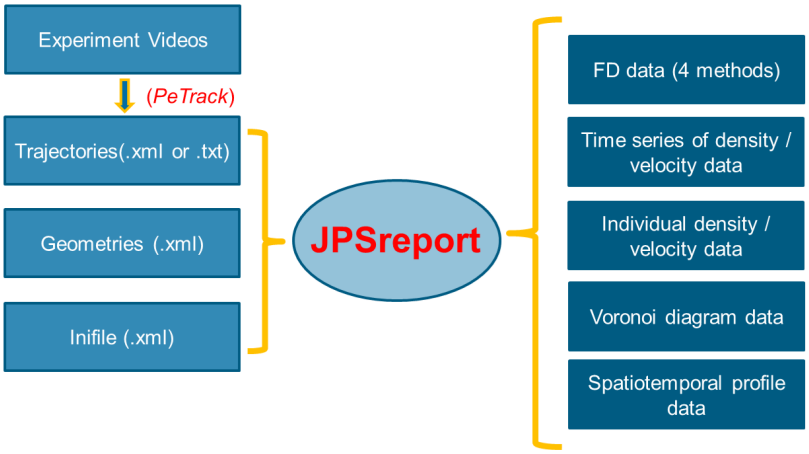


Figure 5.1: Schematic diagram of JPSreport.

The current version requires three inputfiles:

- project file with information about the measurement areas and methods.
- trajectories file containing the pedestrians position over time (see section 3.5)
- geometry file (see section 3.4).

The output files including the information of flow, density and speed are stored in different folders as plain text in ASCII format.

5.1 Building

The only dependency is the open-source Boost geometrical library (www.boost.org) used for generating the Voronoi diagrams. For additional build instruction, please consult the wiki at https://cst.version.fz-juelich.de/jupedsim/jpsreport/wikis/home.

5.2 Input files

A program call looks like this:

```
> ./JPSreport.exe inifile.xml
```

where $inifile.xml$ contains all the information needed to performs the analysis.

5.2.1 Configuration file

The minimum contain of the initialization file is presented in Listing 5.1.

Listing 5.1: Structure of a project file

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

<JPSreport project="JPS-Project" version="0.7" xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance" xsi:noNamespaceSchemaLocation="../../xsd/jps_report.xsd"
    >

    <!-- geometry file -->
    <geometry file = "./Testing/geo_BOT360.xml" />
    <!-- trajectories file and format -->
    <!-- either a file name or a path location. In the latter case all files in
        the directory will be used-->
    <trajectories format="xml", unit="m">
        <file name="bot-360-160-160.xml" />
        <file name="bot-360-120-120.xml" />
        <path location="../JPSdata/data/BO/BOT360/" />
    </trajectories>
    <!--give relative path of scripts based on the location of inifile or give
        the absolute path-->
    <scripts location="../scripts/"/>
    <measurement_areas unit="m">
        <area_B id="1" type="BoundingBox">
            <vertex x="-4.00" y="0" /> <!-- CCW -->
            <vertex x="-4.00" y="3.60" />
            <vertex x="4.00" y="3.60" />
            <vertex x="4.00" y="0" />
            <length_in_movement_direction distance="1.0" />
        </area_B>
        <area_B id="2" type="BoundingBox">
            <vertex x="-5.00" y="0" /> <!-- CCW -->
            <vertex x="-5.00" y="3.60" />
```

```
            <vertex x="3.00" y="3.60" />
            <vertex x="3.00" y="0" />
            <length_in_movement_direction distance="1.0" />
        </area_B>
        <area_L id="3" type="Line">
            <start x="4.00" y="0" />
            <end x="4.00" y="3.60" />
        </area_L>
        <area_L id="4" type="Line">
            <start x="0.00" y="0" />
            <end x="4.00" y="0" />
        </area_L>
    </measurement_areas>
    <velocity>
        <use_x_component>true</use_x_component>
        <use_y_component>false</use_y_component>

        <!-- frame_step (deltaF) in [frame] that used to calculate instantaneous
             velocity of ped i here v_i = (X(t+deltaF/2) - X(t-deltaF/2))/(deltaF
             ). X is location. -->
        <frame_step>5</frame_step>
    </velocity>
    <!-- Method A (Zhang2011a) Flow and Vel -->
    <method_A enabled="false">
        <!-- frame_interval used to count the flow [fr] -->
        <frame_interval unit="frame">
            70
        </frame_interval>
        <!-- The coordinate of the line used to calculate the flow and velocity
             -->
        <measurement_area id="3" />
        <measurement_area id="4" />
    </method_A>
    <!-- Method B (Zhang2011a) Vel and Dens based on Tin and Tout -->
    <method_B enabled="false">
        <measurement_area id="1" />
    </method_B>
    <!-- Method C (Zhang2011a) Classical density and Vel -->
    <method_C enabled="false">
        <measurement_area id="1" />
        <measurement_area id="2" />
    </method_C>
    <!-- Method D (Zhang2011a) Voronoi density and Vel -->
    <method_D enabled="true" output_graph="false" individual_FD="true">
        <measurement_area id="1" />
        <measurement_area id="2" />
        <cut_by_circle enabled="false" radius ="0.5" edges ="6"/>
        <profiles enabled="false" grid_size_x="0.10" grid_size_y="0.10"/>
    </method_D>
</JPSreport>
```

trajectories

This option specifies the format, name and location of the input trajectories. If a "file name" is given, then only the assigned file will be analysed. Otherwise, all the files with the given format in the assigned location will be analysed. Note that, 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 "/" or contains ":".

measurement_areas
There are two different types of measurement areas corresponding to the different measurement methods. Several measurement areas (polygons) with different Id number can be supplied. The types are:

- ⟨**BoundingBox**⟩ Bounding box is a polygon. The vertices of the box should be clockwise.
- ⟨**Line**⟩ The line is only used in Method A (see below).

Velocity

- If only ⟨**use_x_component**⟩ is "**true**", the velocity in x direction is used in analysis.
- If only ⟨**use_y_component**⟩ is "**true**", the velocity in y direction is used in analysis.
- If both ⟨**use_x_component**⟩ and ⟨**use_y_component**⟩ are "**true**", the velocity in 2D space is used in analysis.
- ⟨**frame_step**⟩is the number of frames ($\Delta t'$) used to calculate the instantaneous velocity. Since the instantaneous velocity is calculated according to

$$v_i(t) = \frac{x_i(t + \Delta t'/2) - x_i(t - \Delta t'/2))}{\Delta t'} \tag{5.1}$$

  Note that at least one of the above variables should be "true" during the analysis.

Measurement Methods (A, B, C, D): details refer to Section 5.4

1. ⟨**Method A**⟩
   Parameters:
   a) Location of reference line (Start point and End point)
   b) Value of time interval (frame) for calculating flow rate

   Output data:
   File 1: N-t (the accumulative number of pedestrians passing the reference line and the corresponding time in frame)
   File 2: the mean flow and velocity over the given time interval.

   Note that: All the obtained data is based on the same reference line.

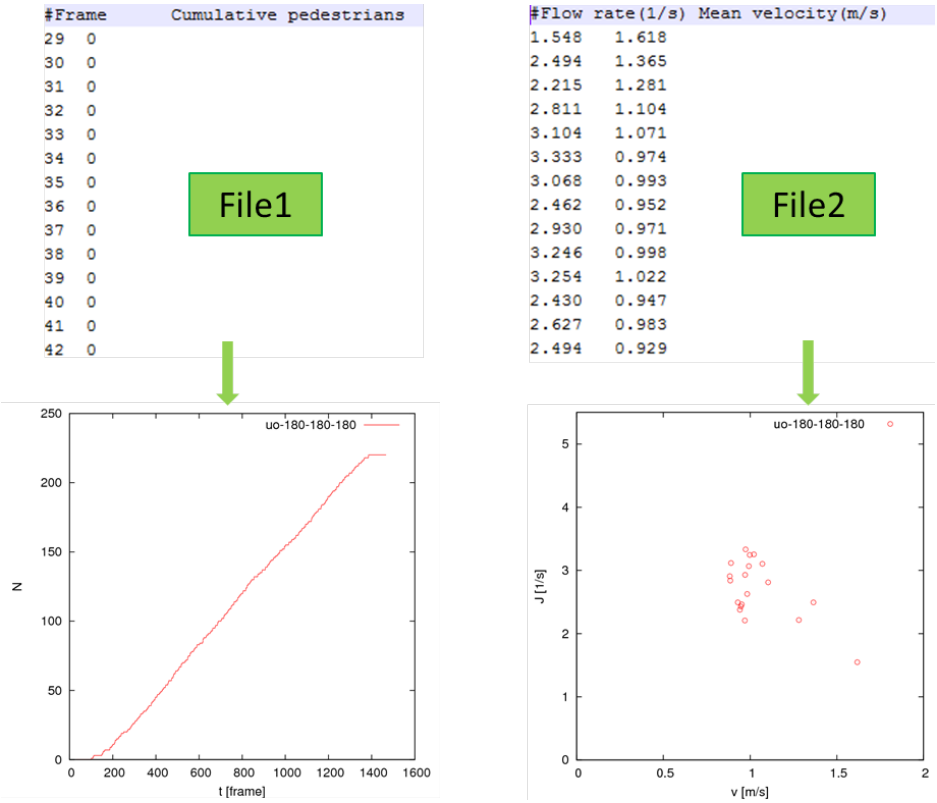


Figure 5.2: Output data from Method A.

2. ⟨**Method B**⟩
   Method B can only be used to analyse one directional pedestrian movement in a corridor. The speed is defined by the length of the measurement area and the time a pedestrian stay in the area.
   Parameters:
   a) measurement_area (should be rectangle)
   b) length_in_movement_direction

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

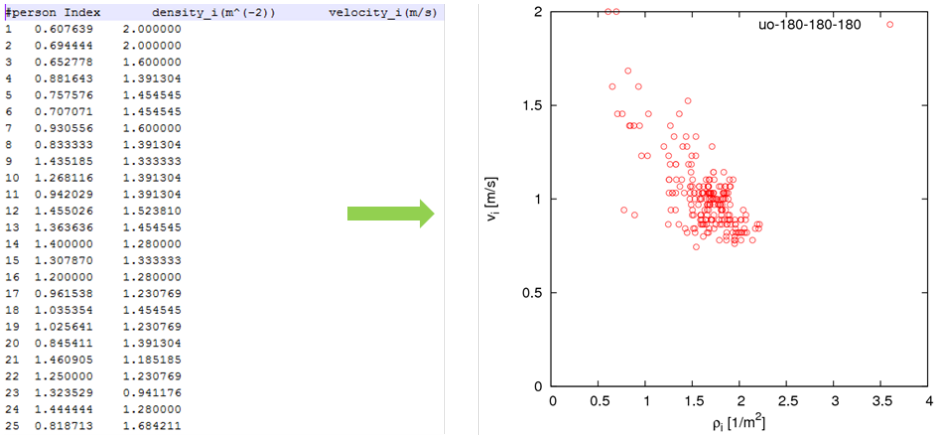


Figure 5.3: Output data from Method B.

3. ⟨**Method C**⟩
   Parameters: measurement area
   Output data: mean density and velocity over time ($\rho_C(t)$ and $v_C(t)$).

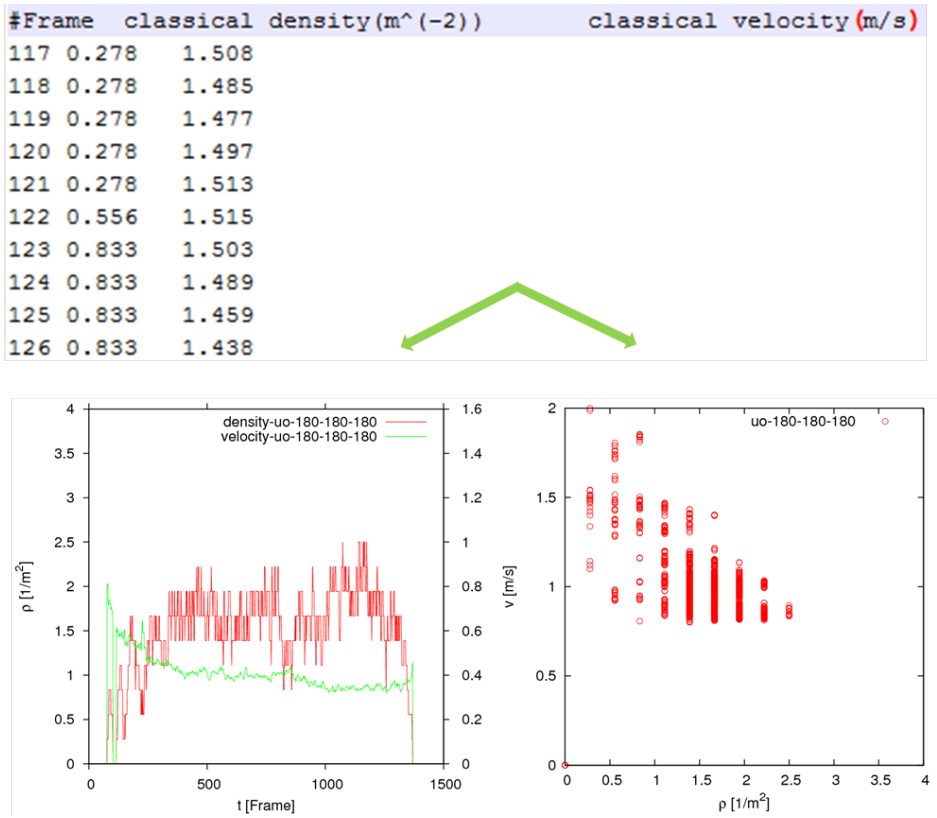


Figure 5.4: Output data from Method C.

4. ⟨**Method D**⟩

   Parameters:

   - ⟨**measurement_area**⟩: the id of the measurement area.
   - ⟨**output_graph**⟩: determines whether or not to output data for visualizing the Voronoi diagram. If true, files including Voronoi cells, speed and the coordinates of pedestrian corresponding to each cell will be written in the folder (*Output/Fundamental_Diagram/Classical_Voronoi/VoronoiCell*).
   - ⟨**cut_by_circle**⟩: determines whether or not to cut each cell by circle. "radius" of the circle and the number of "edges" for approximating the circle should be supplied if this is chosen.
   - ⟨**individual_FD**⟩: determines whether or not to output the data for individual fundamental diagram, which is based on the Voronoi density and velocity and each pedestrian but not averaged value over space. If true, the related data will be written in the folder (*Output/Fundamental_Diagram/IndividualFD*).
   - ⟨**profiles**⟩: determines whether to calculate the profiles over time and space. If yes, the resolution which is decided by the parameters '⟨**grid_size_x**⟩' and '⟨**grid_size_y**⟩' should be set. The data will be in the folder (*Output/Fundamental_Diagram/Classical_Voronoi/field/*).

   Output data: mean density and velocity over time ($\rho_V(t)$ and $v_V(t)$). Sample data for plotting the Voronoi cells are presented in Figure 5.7. Data for plotting profiles are in Figure 5.8. Data of individual Fundamental diagram are in Figure 5.5.

#Frame	Voronoi density(m^(-2))	Voronoi velocity(m/s)
117	0.250	1.513
118	0.261	1.487
119	0.274	1.477
120	0.287	1.494
121	0.301	1.507
122	0.315	1.506
123	0.325	1.493

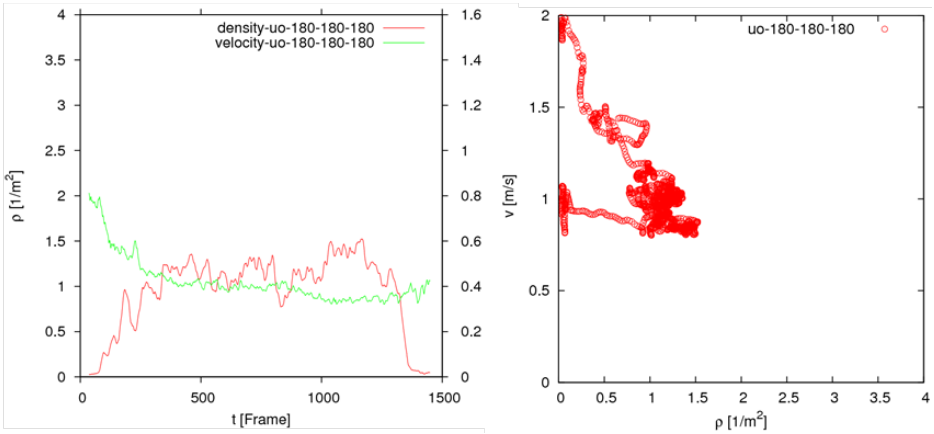


Figure 5.5: Output data from Method D: Voronoi density and velocity.

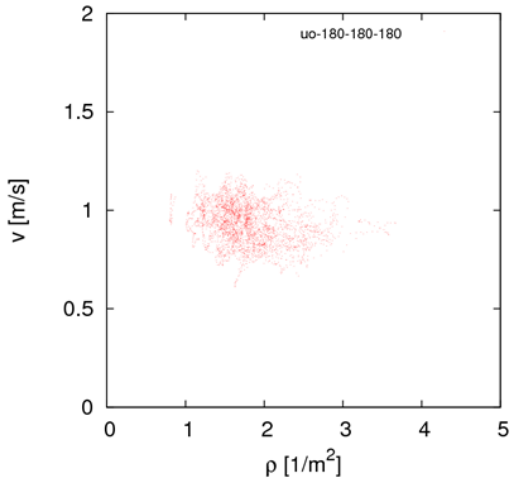


Figure 5.6: Output data from Method D: Individual fundamental diagram.

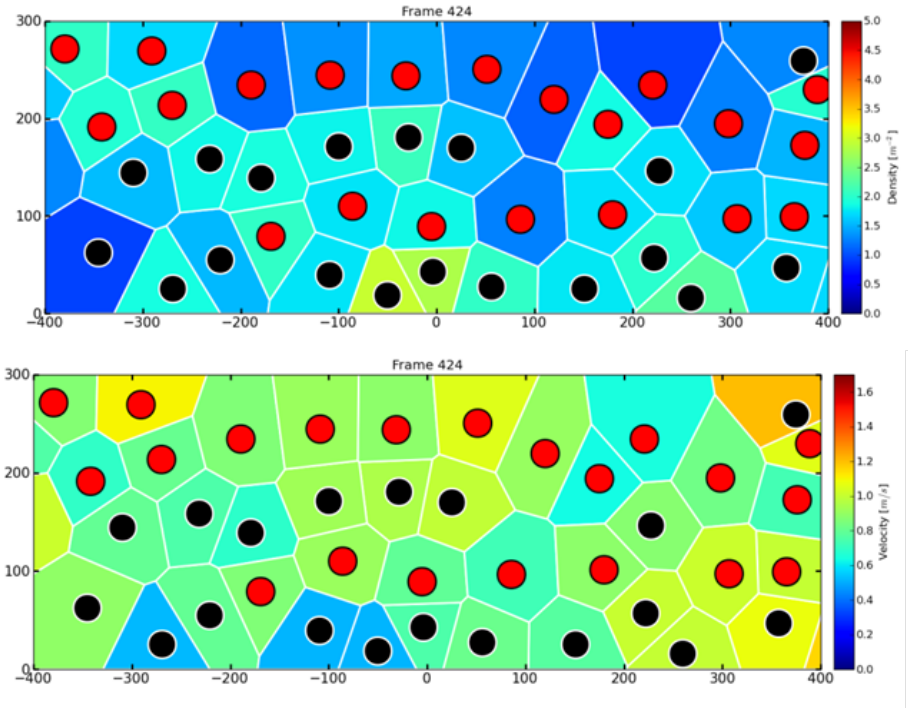


Figure 5.7: Output data from Method D: Voronoi cells.

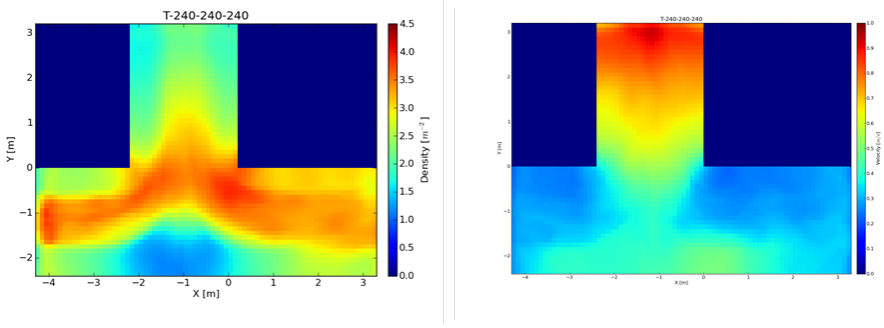


Figure 5.8: Output data from Method D: Voronoi profiles.

5.2.2 Geometry file

In JPSreport the geometry is regarded as an unique polygon. Obstacles also defined as polygons are allowed. Attention should be paid to the following points while creating the file:

- The polygon has to be **closed**. In other words, the first and the last vertex data should be the same and entrances or exits are not considered.

- The vertex data for the polygon should be written in order (either in clockwise or anticlockwise).
- The built polygon should contain all pedestrians during the whole experiment. Any pedestrian outside the geometry at any time could lead to wrong results.
- The unit should be m.

For example, The geometry file of the scenario in Figure 5.9 can be written as following:

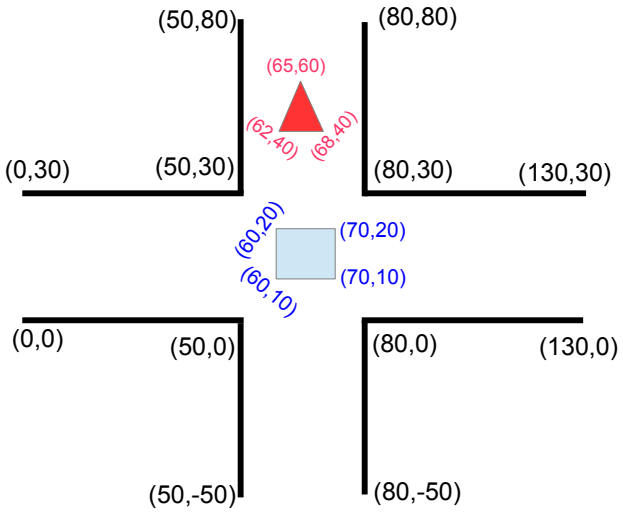


Figure 5.9: Sample geometry.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<geometry version ="0.5" caption="crossing" unit="m">
    <rooms>
        <room id="0" caption="hall" >
            <subroom id="0" class="subroom">
                <polygon caption="wall">
                    <vertex px="0.0" py="0.0"/>
                    <vertex px="0.0" py="30.0"/>
                    <vertex px="50.0" py="30.0"/>
                    <vertex px="50.0" py="80.0"/>
                    <vertex px="80.0" py="80.0"/>
                    <vertex px="80.0" py="30.0"/>
                    <vertex px="130.0" py="30.0"/>
                    <vertex px="130.0" py="0.0"/>
                    <vertex px="80.0" py="0.0"/>
                    <vertex px="80.0" py="-50.0"/>
                    <vertex px="50.0" py="-50.0"/>
                    <vertex px="50.0" py="0.0"/>
```

```
                    <vertex px="0.0" py="0.0"/>
                </polygon>
                <obstacle id="1" caption="table" height="1.0" >
                    <polygon>
                        <vertex px="60.0" py="10.0"/>
                        <vertex px="60.0" py="20.0"/>
                        <vertex px="70.0" py="20.0"/>
                        <vertex px="70.0" py="10.0"/>
                        <vertex px="60.0" py="10.0"/>
                    </polygon>
                </obstacle>
                <obstacle id="2" caption="table" height="1.0" >
                    <polygon>
                        <vertex px="62.0" py="40.0"/>
                        <vertex px="65.0" py="60.0"/>
                        <vertex px="68.0" py="40.0"/>
                        <vertex px="62.0" py="40.0"/>
                    </polygon>
                </obstacle>
            </subroom>
        </room>
    </rooms>
</geometry>
```

5.2.3 Trajectory file

JPSreport supports the formats *.xml* and *.txt* defined in section 3.5.

There are python scripts (*txt2xml.py* and *txt2xml_lib.py*) in the 'scripts' folder which can be used transfer trajectory data from *.txt* file exported from *PeTrack*. The script is called as

```
$ run txt2xml.py -s 0 -f 16
```

To get more information on the commands run:

```
$ run txt2xml.py -h
```

5.3 Limitations

There are still a few limitations in the current version of JPSreport:

- Not able to analyze trajectories directly from periodic boundary conditions. When a pedestrian enters the scenario for a second time, a new ID should be assigned.
- Not able to analyze data at complex geometries with more than one polygon.
- Not able to automatically determine the steady state.

5.4 Methodologies

JPSreport is based on the following four measurement methods referred in [?]:

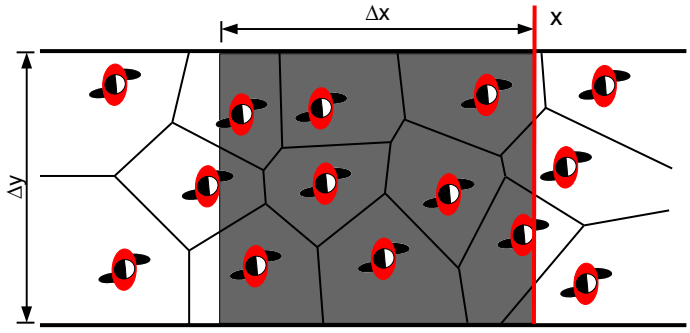


Figure 5.10: Illustration of different measurement methods. $Method\ A$ is a kind of local measurement at cross-section with position x averaged over a time interval Δt, while $Methods\ B-D$ measure at a certain time and average the results over space Δx. Note that for $Method\ D$, the Voronoi diagrams are generated according to the spatial distributions of pedestrians at a certain time.

5.4.1 Method A

For $MethodA$, a reference location x in the corridor is taken and studied over a fixed period of time Δt (as shown in Figure 5.10). Mean values of flow and velocity are calculated over time. We refer to this average by $\langle\rangle_{\Delta t}$. Using this method we obtain the time t_i and the velocity v_i of each pedestrian passing x 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

$$\langle J\rangle_{\Delta t} = \frac{N_{\Delta t}}{t_{N_{\Delta t}} - t_{1_{\Delta t}}} \qquad \text{and} \qquad \langle v\rangle_{\Delta t} = \frac{1}{N_{\Delta t}} \sum_{i=1}^{N_{\Delta t}} v_i(t) \tag{5.2}$$

where $N_{\Delta t}$ is the number of persons passing the location x during the time interval Δt. $t_{1_{\Delta t}}$ and $t_{N_{\Delta t}}$ are the times when the first and last pedestrians pass the location in Δt. They could be different from Δt. The time mean velocity $\langle v\rangle_{\Delta t}$ is defined as the mean value of the instantaneous velocities $v_i(t)$ of the $N_{\Delta t}$ persons according to equation (5.3). We calculate $v_i(t)$ by use of the displacement of pedestrian i in a small time interval $\Delta t'$ around t:

$$v_i(t) = \frac{\|\vec{x_i}(t + \Delta t'/2) - \vec{x_i}(t - \Delta t'/2)\|}{\Delta t'} \tag{5.3}$$

5.4.2 Method B

The second 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 with length Δx in the 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 (see equation (A.6)),

$$\langle v \rangle_i = \frac{\Delta x}{t_{i,\text{out}} - t_{i,\text{in}}} \tag{5.4}$$

where $t_{i,\text{in}}$ and $t_{i,\text{out}}$ are the times a person i enters and exits the measurement area, respectively. The density ρ_i for each person is calculated with equation (5.5):

$$\langle \rho \rangle_i = \frac{1}{t_{i,\text{out}} - t_{i,\text{in}}} \cdot \int_{t_{i,\text{in}}}^{t_{i,\text{out}}} \frac{N'(t)}{\Delta x \cdot \Delta y} dt \tag{5.5}$$

Δy is the width of the measurement area while $N'(t)$ is the number of person in this area at a time t.

5.4.3 Method C

The third measurement method, we call it classical method. The density $\langle \rho \rangle_{\Delta x}$ is defined as the number of pedestrians divided by the area of the measurement section:

$$\langle \rho \rangle_{\Delta x} = \frac{N}{\Delta x \cdot \Delta y} \tag{5.6}$$

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

$$\langle v \rangle_{\Delta x} = \frac{1}{N} \sum_{i=1}^{N} v_i(t) \tag{5.7}$$

5.4.4 Method D

This method is based on Voronoi diagrams [12] 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 diagrams (see Figure 5.10) 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} (see Figure **??**) are defined as

$$\rho_{xy} = 1/A_i \quad \textit{and} \quad v_{xy} = v_i(t) \qquad \text{if } (x, y) \in A_i \tag{5.8}$$

where $v_i(t)$ is the instantaneous velocity of each person, see equation (5.3). The Voronoi density and velocity for the measurement area is then defined as [11]

$$\langle \rho \rangle_v = \frac{\iint \rho_{xy} dx dy}{\Delta x \cdot \Delta y} \tag{5.9}$$

$$\langle v \rangle_v = \frac{\iint v_{xy} dx dy}{\Delta x \cdot \Delta y} \tag{5.10}$$

5.5 Detection of steady state

Experiments with pedestrians could depend strongly on initial conditions. Comparisons of the results of such experiments require to distinguish carefully between transient state and steady state. It is worth noting that pedestrian movement includes transient state and steady state. In pedestrian dynamics experiments, transient state depends strongly on initial conditions while steady state is a good indicator of the independency of initial conditions, especially when the duration of experiments is short. Thus steady state is a distinguished significance for the interpretation of the results, and transient state should be excluded when combining different experiments to get universal conclusions.

5.5.1 Detection process

The detection is not yet fully automated and still need some input from the user in terms of the reference processes. The reference processes are selected with the relatively stable states in density and velocity (see the interval between the two green lines in) The detection of steady state is based on the output data from Method D: Voronoi density and velocity (see Figure 5.5). A program call looks like this:

```
> python SteadyState.py -n ../input/InputFile.dat -rs 240
  -re 640 -vs 240 -ve 640 -f yes
```

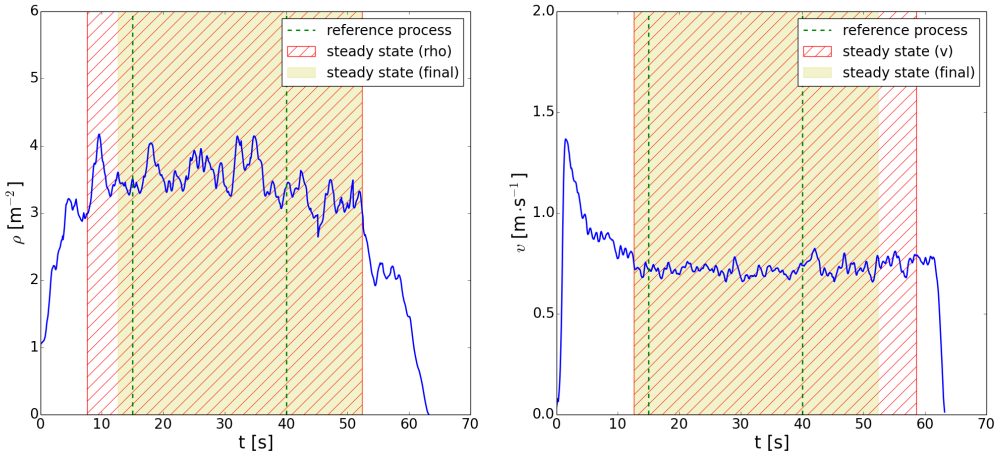
where

- **-n** the input file
- **-rs** start of the reference for the density
- **-re** end of the reference for the density
- **-vs** start of the reference for the velocity

- **-ve** end of the reference for the velocity

To get the full documentation, run:

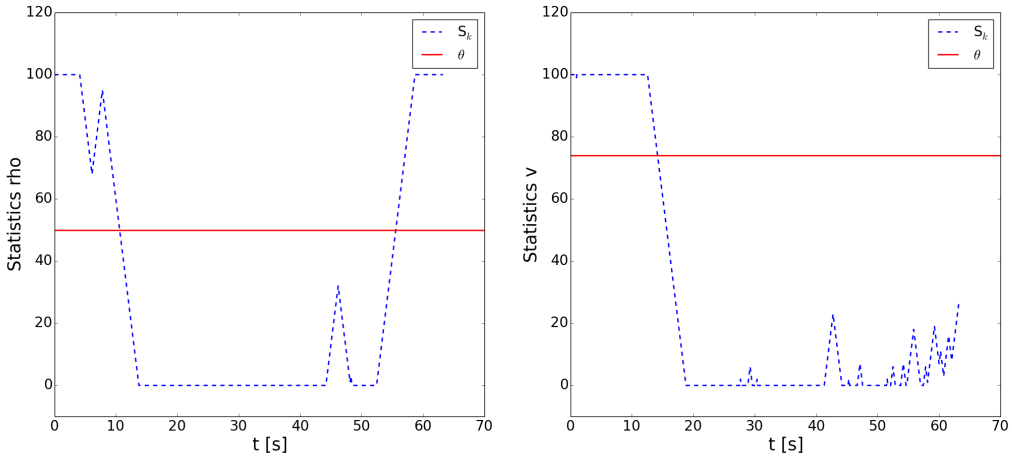
```
> python SteadyState.py -h
```



(a) Density (b) Velocity

Figure 5.11: Time series of input data.

A modified Cumulative Sum Control Chart algorithm is used to calculate cusum statistics, and an autoregressive model is used to calibrate the threshold of the detection parameter [7]. The detected steady state is using the intersection points of the statistics and the threshold minus the reaction time (see Figure 5.12). The corresponding detected steady state is shown in Figure 5.11 by the interval with the red slashed area. The final steady state is the intersection part of the steady states in density and velocity (see the interval with the yellow filled area in Figure 5.11). The start frame and the end frame of the detected steady states and the final steady state are shown in the output files (see Figure 5.13).



(a) Density (b) Velocity

Figure 5.12: Detection of steady state.

```
# start end
202 838
```

Figure 5.13: Output data from steady state detection.

5.5.2 Limitations

There are still a few limitations in the current version of Steady State Detection:

- Not able to select reference processes automatically.
- Not able to deal with multiple steady states within the same input data.

Chapter 6

JPSvis

JPSvis is the tool used for visualizing the trajectories. It is available on Linux, Windows and OS X platforms. It features a simple and streamlined user interface build with Qt and VTK.

6.1 Pre-compiled binaries

Pre compiled binaries are available from our official page. After downloading and unpacking the archived, you can drag and drop a trajectory file or just the geometry file on the executable.

6.2 Compiling from sources

The packages needed for compiling are:

1. Qt Toolkit, version 4.5.3 or newer available at http://qt-project.org/.
2. VTK: Visualization Toolkit, version 5.4 or newer available from http://www.vtk.org/VTK/resources/software.html.
3. FFMPEG: Needed for video support under Linux.

Download and unpack the sources from https://github.com/JuPedSim/JuPedSim. You may need to adjust the path to vtk in the JPSvis.pro file. Then run

```
> qmake JPSvis.pro
> make
```

Chapter 7

JPSeditor

JPSeditor is the geometry editor provided with the JuPedSim bundle. It features a direct import and export of dxf files. It is written in C++ using the Qt libraries. A video tutorial is available on YouTube channel.

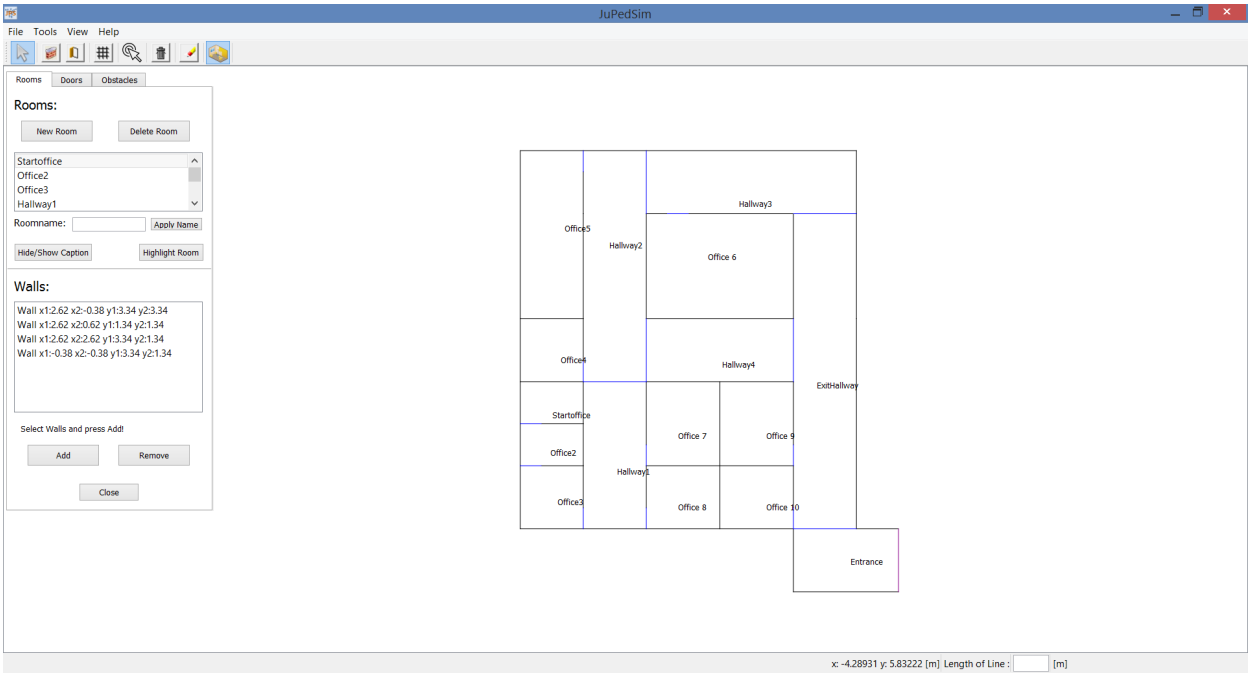


Figure 7.1: Editor for geometry files with dxf import capabilities.

Bibliography

[1] Guidelines for evacuation analysis for new and existing passenger ships, Oct 2007.

[2] M. Chraibi, Seyfried A. Kemloh, U., and A. Schadschneider. Force-based models of pedestrian dynamics. Networks and Heterogeneous Media, 6(3):425–442, 2011.

[3] Mohcine Chraibi, Armin Seyfried, and Andreas Schadschneider. Generalized centrifugal force model for pedestrian dynamics. Physical Review E, 82:046111, 2010.

[4] David Haensel. A knowledge-based routing framework for pedestrian dynamics simulation. Master's thesis, Technische Universitaet Dresden, November 2014.

[5] Armel Ulrich Kemloh Wagoum, Armin Seyfried, and Stefan Holl. Modeling the dynamic route choice of pedestrians to assess the criticality of building evacuation. Advances in Complex Systems, 15(3), 2012.

[6] Armel Ulrich Kemloh Wagoum, Bernhard Steffen, Armin Seyfried, and Mohcine Chraibi. Parallel real time computation of large scale pedestrian evacuations. Advances in Engineering Software, 60-61:98–103, 2013. CIVIL-COMP: Parallel, Distributed, Grid and Cloud Computing.

[7] Weichen Liao, Antoine Tordeux, Armin Seyfried, Mohcine Chraibi, Kevin Drzycimski, Xiaoping Zheng, and Ying Zhao. Steady state of pedestrian flow in bottleneck experiments. arXiv, 2015.

[8] Andrea Portz and Armin Seyfried. Analyzing stop-and-go waves by experiment and modeling. In R.D. Peacock, E.D. Kuligowski, and J.D. Averill, editors, Pedestrian and Evacuation Dynamics 2010, pages 577–586. Springer, 2011.

[9] Rimea-richtlinie für mikroskopische entfluchtungs-analysen, 2007. www.rimea.de.

[10] E. Ronchi, S. M. V. Gwynne, D. A. Purser, and P. Colonna. Representation of the impact of smoke on agent walking speeds in evacuation models. Fire Technology, 49(2):411–431, 2013.

[11] B. Steffen and A. Seyfried. Methods for measuring pedestrian density, flow, speed and direction with minimal scatter. Physica A, 389(9):1902–1910, may 2010.

[12] G. M. Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Journal für die reine und angewandte Mathematik, 133:97–178, 1908.

Appendices

Appendix A

The Gompertz Model

A.1 Model definition

The Gompertz model is based on a continuous "physical" force and shares many parameters with the Generalized Centrifugal Force Model. Both models in fact only differ in the definition of the repulsive force (see Equation A.1 and Equation A.3).

Given the Gompertz function:

$$\mathcal{G}(\beta_{ij}) = \exp\big(-b\exp(-c\beta_{ij})\big), \tag{A.1}$$

with

- b the displacement along the x-axis $\equiv$ cut off radius
- c the growth rate (y scaling).

The function (A.1) depends on the quantity:

$$\beta_{ij} = 1 - \frac{d_{ij}}{r_i + r_j}, \tag{A.2}$$

whit d_{ij} the distance between pedestrians i with radius r_i and j with radius r_j.

The repulsive force is then defined as:

$$\overrightarrow{F_{ij}}^{\text{rep}} = -\eta' \cdot \parallel \overrightarrow{v_i^0} \parallel \cdot \mathcal{G}(\beta_{ij}) \cdot \overrightarrow{e_{ij}}, \tag{A.3}$$

A.2 Cutoff radius

While the cutoff radius for the GCFM is an input parameter, in the Gompertz model we have to derive it from other parameters.

For this porpoise we choose a cutoff radius β_c such that, the distance (between centres) is equal to a multiple of the sum of radii,

$$\mathcal{G}(\beta < \beta_c) \ll 0.$$

Considering Eq. (A.2) we set:

$$\beta_c = -2.$$

(corresponding to $3\times$ the sum of radii)

Remark:

$\beta < \beta_c$ and $d - r_1 - r_2 = -(r_1 + r_2)\beta$ imply

$$d - r_1 - r_2 \geq \underbrace{-2\mathrm{max}(\mathrm{E_a}, \mathrm{E_b})\beta_\mathrm{c}}_{\text{cutoff}}. \tag{A.4}$$

E_a and E_b are resp. the semi-axes in the direction of motion and orthogonal to it.

We have

$$\max(E_a) = a_{\min} + a_\tau v^0$$

and

$$\max(E_b) = 0.5(b_{\min} + 0.49)$$

For $a_{\min} = 0.18$ m, $a_\tau = 0.23$ s and $v^0 = 1.34$ m/s we get:

$$\max(E_a) = 0.488\ m$$

And for $b_{\min} = 0.4$ m we get:

$$\max(E_b) = 0.445\ m$$

These values imply for Eq. (A.4) that the cutoff radius for the effective is about 2 m (0.488· 2· 2 = 1.952).

A.3 Model Calibration

We want to choose b and c such that the following inequalities are fulfilled

$$\mathcal{G}(\beta < \beta_c) < \epsilon \tag{A.5}$$

and

$$\mathcal{G}(\beta > \beta_m) > \Theta, \tag{A.6}$$

β_c being the cutoff radius and β_m the maximum allowed β_{ij} reducing the amount of possible overlapping between pedestrians i and j. Θ is a threshold and ϵ is a small constant.

From Eqs. (A.5) and (A.6) we have:

$$\begin{cases} \exp\big(-b\exp(-c\beta_c)\big) = \epsilon \rightarrow -b\exp(-c\beta_c) = \log(\epsilon) & : (E) \\ \exp\big(-b\exp(-c\beta_m)\big) = \Theta \rightarrow -b\exp(-c\beta_m) = \log(\Theta) & : (F) \end{cases}$$

(E)/(F):

$$\exp(-c\beta_c) \cdot \exp(c\beta_m) = K,$$

with $K = \log(\epsilon)/\log(\Theta)$.

$$\exp(-c(\beta_c - \beta_m)) = K,$$

$$c = \frac{\log(K)}{\beta_m - \beta_c}.$$

(E)+(F):

$$-b\big(\exp(-c\beta_c) + \exp(-c\beta_m)\big) = \log(\epsilon) + \log(\Theta).$$

$$b = -\frac{\log(\epsilon \cdot \Theta)}{\exp(-c\beta_c) + \exp(-c\beta_m)}.$$

Simple calculation yields:

```
>> b=0.034229, c=2.450932
```

See Figure A.1 for the shape of the repulsive force using the aforementioned calculation of b and c.

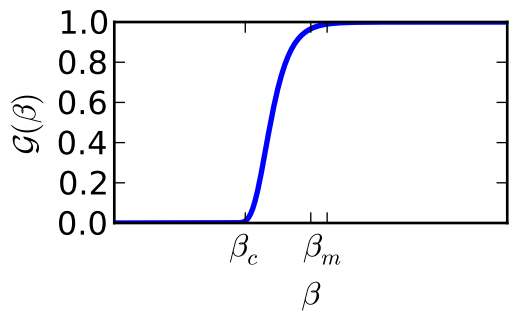
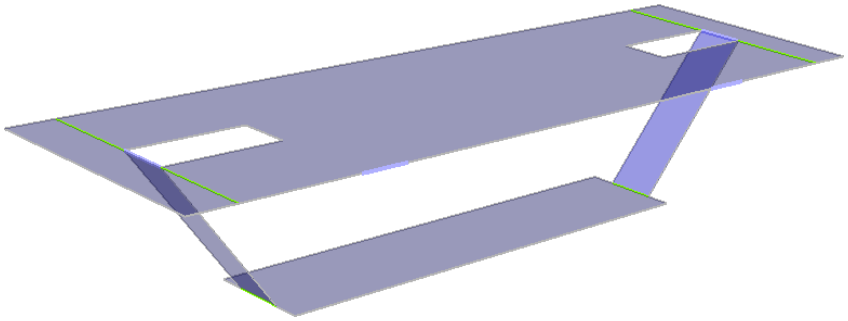


Figure A.1: The repulsive force with respect to β for b=0.034229, c=2.450932.

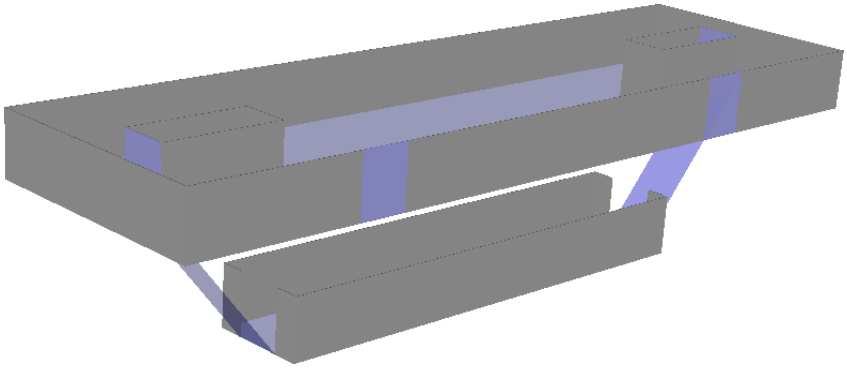
Appendix B

Showcases

B.1 Stairs geometry



(a) 3D view of the floor



(b) 3D view displaying walls

Figure B.1: Sample geometry with stairs.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<geometry version="0.5" caption="Room with two stairs">
    <rooms>
        <room id="0" caption="subway">
            <subroom id="0" class="subroom" A_x="0" B_y="0" C="0">
                <polygon caption="wall">
                    <vertex px="0.0" py="1.0" />
                    <vertex px="0.0" py="0.0" />
                    <vertex px="20.0" py="0.0" />
                    <vertex px="20.0" py="1.0" />
                </polygon>
                <polygon caption="wall">
                    <vertex px="20.0" py="3.0" />
                    <vertex px="20.0" py="4.0" />
                    <vertex px="0.0" py="4.0" />
                    <vertex px="0.0" py="3.0" />
                </polygon>
            </subroom>
            <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>
            <subroom id="2" class="stair" A_x="1.2" B_y="0" C="-24">
                <polygon caption="wall">
                    <vertex px="20.0" py="1.0" />
                    <vertex px="25.0" py="1.0" />
                </polygon>
                <polygon caption="wall">
                    <vertex px="20.0" py="3.0" />
                    <vertex px="25.0" py="3.0" />
                </polygon>
                <up px="25.0" py="2" />
                <down px="20.0" py="2" />
            </subroom>
            <crossings>
                <crossing id="0" subroom1_id="0" subroom2_id="1">
                    <vertex px="0.0" py="1.0" />
                    <vertex px="0.0" py="3.0" />
                </crossing>
                <crossing id="1" subroom1_id="0" subroom2_id="2">
                    <vertex px="20.0" py="1.0" />
                    <vertex px="20.0 " py="3.0" />
                </crossing>
            </crossings>
        </room>
        <room id="1" caption="Hall">
            <subroom id="0" class="subroom" A_x="0" B_y="0" C="6">
                <polygon caption="wall">
                    <vertex px="-5.0" py="7.0" />
                    <vertex px="25.0" py="7.0" />
                </polygon>
                <polygon caption="wall">
                    <vertex px="2.0" py="-3.0" />
                    <vertex px="18.0" py="-3.0" />
```

```
                </polygon>
                <polygon caption="wall">
                    <vertex px="-5.0" py="-3.0" />
                    <vertex px="0.0" py="-3.0" />
                </polygon>
                <polygon caption="wall">
                    <vertex px="20.0" py="-3.0" />
                    <vertex px="25.0" py="-3.0" />
                </polygon>
                <polygon caption="wall">
                    <vertex px="25.0" py="1.0" />
                    <vertex px="20.0" py="1.0" />
                    <vertex px="20.0" py="3.0" />
                    <vertex px="25.0" py="3.0" />
                </polygon>
                <polygon caption="wall">
                    <vertex px="-5.0" py="1.0" />
                    <vertex px="0.0" py="1.0" />
                    <vertex px="0.0" py="3.0" />
                    <vertex px="-5.0" py="3.0" />
                </polygon>
            </subroom>
            <subroom id="1" class="subroom" A_x="0" B_y="0" C="6">
                <polygon caption="wall">
                    <vertex px="-5.0" py="-3.0" />
                    <vertex px="-7.0" py="-3.0" />
                    <vertex px="-7.0" py="7.0" />
                    <vertex px="-5.0" py="7.0" />
                </polygon>
            </subroom>
            <subroom id="2"  class="subroom" A_x="0" B_y="0" C="6">
                <polygon caption="wall">
                    <vertex px="25.0" py="-3.0" />
                    <vertex px="27.0" py="-3.0" />
                    <vertex px="27.0" py="7.0" />
                    <vertex px="25.0" py="7.0" />
                </polygon>
            </subroom>
            <crossings>
                <crossing id="2" subroom1_id="0" subroom2_id="2">
                    <vertex px="25.0" py="-3.0" />
                    <vertex px="25.0" py="1.0" />
                </crossing>
                <crossing id="3" subroom1_id="0" subroom2_id="2">
                    <vertex px="25.0" py="3.0" />
                    <vertex px="25.0 " py="7.0" />
                </crossing>
                <crossing id="4" subroom1_id="0" subroom2_id="1">
                    <vertex px="-5.0" py="-3.0" />
                    <vertex px="-5.0 " py="1.0" />
                </crossing>
                <crossing id="5" subroom1_id="0" subroom2_id="1">
                    <vertex px="-5.0" py="3.0" />
                    <vertex px="-5.0 " py="7.0" />
                </crossing>
            </crossings>
        </room>
    </rooms>

    <transitions>
        <transition id="6" caption="No_Name 1" type="emergency" room1_id="0"
              subroom1_id="1" room2_id="1" subroom2_id="1">
```

```
124             <vertex px="-5.0" py="1.0" />
125             <vertex px="-5.0" py="3.0" />
126         </transition>
127         <transition id="7" caption="No_Name 2" type="emergency" room1_id="0"
                subroom1_id="2" room2_id="1" subroom2_id="2">
128             <vertex px="25.0" py="1.0" />
129             <vertex px="25.0" py="3.0" />
130         </transition>
131         <transition id="8" caption="No_Name 3" type="emergency" room1_id="1"
                subroom1_id="0" room2_id="-1" subroom2_id="-1">
132             <vertex px="0.0" py="-3.0" />
133             <vertex px="2.0" py="-3.0" />
134         </transition>
135         <transition id="9" caption="No_Name 4" type="emergency" room1_id="1"
                subroom1_id="0" room2_id="-1" subroom2_id="-1">
136             <vertex px="18.0" py="-3.0" />
137             <vertex px="20.0" py="-3.0" />
138         </transition>
139     </transitions>
140 </geometry>
```

Appendix C

Previous release notes

C.1 Version 0.6 (31.01.2015)

New module (JPSreport)

- New module for analysing trajectories and generating different kind of plots including time series, Voronoi densities, profiles, fundamental diagrams.

JPScore

- Steering the simulation with predefined events (closing or opening doors during the simulation)
- Information sharing between the pedestrians. The agents now share their knowledge about closed doors.
- Pre evacuation time.
- Adjustable velocities on stairs and even terrain.
- Stability and performance improvement. The simulation is now faster and you will notice it.
- New route choice model, cognitive map, giving agents the possibility to explore the environment and discover doors for instance.
- Different sensors for improved navigation (See subsection 3.1.7).
- Statistics about doors usage and area egress time at the end of the simulation.
- New verification tests.
- Support for Visual Studio and Xcode compilers.

JPSvis

- Saving and restoring settings from a previous session.
- Making high quality videos directly from the visualization interface or generating png sequences
- performance improvements.

C.2 Version 0.5 (5 Aug 2014)

We are proud to announce the first pre-release of our software `JuPedSim` for simulating pedestrians evacuations. Please note that it is a pre-release version for developers only. We are working hard towards the final release for this version. Two modules are shipped with this pre-release:

JPScore: command line simulation core

JPSvis: visualization module

Features

- Simulate pedestrians movement in a space continuous geometry
- Forces based models for describing the pedestrians interactions
- Shortest and quickest path route choice strategies
- Loading and visualizing trajectories and geometries
- Easy to use visualization interface
- Making high quality videos directly from the visualization interface or generating png sequences
- XML based input files

Installing

This version comes with no installer. So just download the archive corresponding to your architecture and unzip them and you are ready to go. The binaries are only available for windows at the moment. For other architectures (Linux, OSX) you will need to compile the sources. See the section **Compiling from sources**.

Uninstalling

As JuPedSim comes with no installer, you just need to delete the unzipped directory.

Basic usage

Running

```
>> jpscore.exe my_simulation_ini.xml
```

from the command line (or also dropping the file on the executable) will generate a trajectory file, which you can visualize with JPSvis. You will find some projects samples in the downloaded files and further information in the manual.

Showcase

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

Compiling from sources

You can compile the simulation core for your specific platform with the supplied cmake script. The only requirement is a compiler supporting the new standard c++11.

Windows (tested on Win7 with MinGW 4.8)

```
cmake -G "MinGW Makefiles" CMakeLists.txt
make-mingw32.exe
```

Linux (tested on Ubuntu 14.04 with gcc 4.8)

```
cmake   CMakeLists.txt
make
```

OSX (tested on OSX Maverick with clang 5.1)

```
cmake    CMakeLists.txt
make
```

Note that the OpenMP acceleration might be missing under OSX. For the visualization module (`JPSvis`) at least `Qt` version 4.5 and `VTK` version 4.8 are required. You can download the latest version of `QT` here and the latest version of `VTK` (visualization toolkit) here.

System Requirements

There is no special hardware/software requirements for running `JuPedSim`. The distributed binaries however, are only available for windows at the moment. For compiling from sources you need a compiler with c++11 support is needed for the core. `Qt` version >=4.5 and `VTK` >5.8 are needed for the visualization.

Known issues

- Some verification tests are still failing
- Occasional crashes if the input files are not valid. Make sure to validate you XML input files with the supplied XSD files.

Frequently Asked Questions

1. What is the official page of `JuPedSim`?
    - www.jupedsim.org and the contact is `info at jupedsim.org`. You will find more information on the working group and other tools and experimental pedestrians data we have been collecting over the years.
2. Where is the official repository ?
    - `JuPedSim` is developed at the Forschungszentrum Jlich in Germany and the bleeding edge code is in their intern git repository. At the moment only specific tags are pushed to Github.
3. Is there a manual ?

- Of course, the user's guide is found in the downloaded archive.

4. Are the models validated ?

   - We are actually setting up verification and validation tests. Some verification tests are included in this version but most of them will be available with the next version.

5. How can I contribute to `JuPedSim`?

   - Testing and reporting bugs will be great. If you want to contribute actively to the code, by implementing new models and/or features, you are welcome to do so. Please contact uns per mail at `info at jupedsim.org` so that we can grant you access to the repositories.