Project: Simulator of the pedestrians' dynamics
Writer: Hossein Sadeghi
Date: 26.09.2020

1. Introduction

In this project, the dynamics of pedestrians in a system with two intersecting corridors is simulated. The individual pedestrians' dynamics is modeled with a microscopic point of view. The social force model, as a widely used microscopic method of pedestrian simulation has been employed and the solver is developed in the C++ programming language.

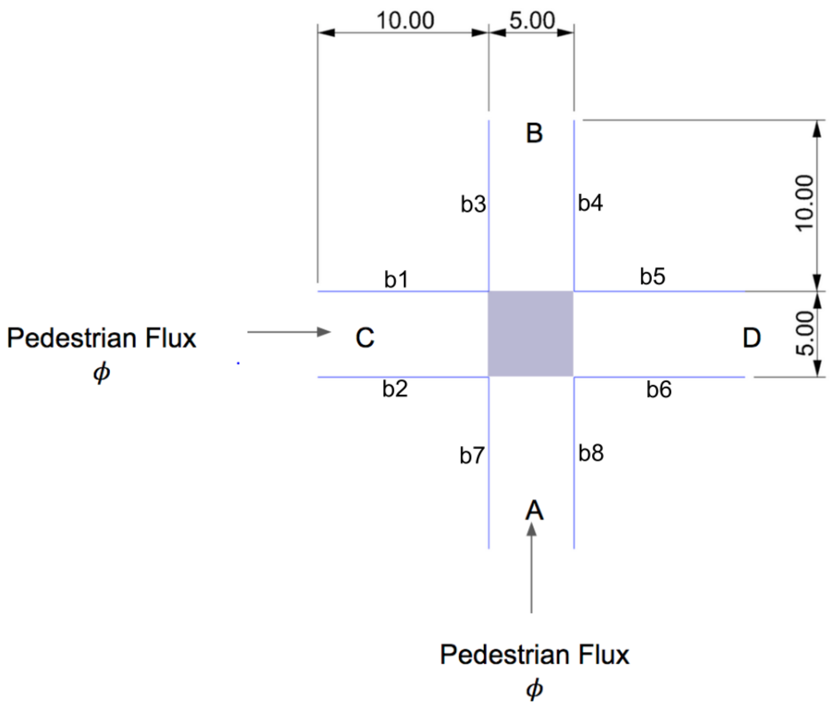

Figure 1: The geometry of two intersecting corridors with the entering gates at A and C and out-gates B and D. The intersection area is defined in blue. b1-8 is the numbering of the walls of the corridors.

2. Theory

For the simulation of pedestrian dynamics, the social force model proposed by Helbing et al. [1] and improved by some others [2] has been employed. This model is a microscopic model and considers individual interactions of pedestrians.

Based on this model, the motion of pedestrian is governed by three forces: driving force, social force and boundary force.

2.1. Driving force

Driving force is the tendency of pedestrian toward the destination, which is defined by an acceleration term acting on pedestrian i:

$$\overrightarrow{F_{di}} = \frac{v_{di}\overrightarrow{e_i} - \overrightarrow{v_i}}{\tau}$$

Where v_{di} is the desired velocity, $\overrightarrow{e_i}$ the desired direction, $\overrightarrow{v_i}$ actual velocity of pedestrian and τ relaxation time of pedestrian to achieve the desired velocity.

2.2. Social force

The interactions between the pedestrian are described by the social force which is a combination of a socio-psychological force (F_{soc}) and a physical or granular force (F_{ph}) important in the panic condition.

$$\overrightarrow{F_{soc}} = A \sum_{j=1, j\neq i}^{N} e^{\frac{(r_i+r_j)-d_{ij}}{B}} \overrightarrow{n_{ji}}.F_{ij}$$

Where A and B are the interaction strength and interaction range between the pedestrians, d_{ij} distance between the pedestrians, r radius of the pedestrian, N total number of the pedestrians, $\overrightarrow{n_{ij}}$ unit vector from j to i and F_{ij} is a form factor considering the anisotropic reaction of pedestrian based on their field of view. This later one will be defined by the angel between the pedestrian and a form factor λ.

The physical force exerted by the other pedestrians is the sum of body and sliding forces.

$$\overrightarrow{F_{ph}} = \sum_{j=1, j\neq i}^{N} k\big[(r_i+r_j)-d_{ij}\big]\overrightarrow{n_{ji}} + \kappa\big[(r_i+r_j)-d_{ij}\big]\Delta v_{ji}^{t}\overrightarrow{t_{ij}}$$

Here $\overrightarrow{t_{ij}}$ and Δv_{ji}^{t} are the tangential direction and velocity difference of the two pedestrians in this direction. k and κ are two large constant values.

2.3. Boundary force

The interaction with the boundaries (walls) are also defined in the same way but with different values of interaction strength and range and without anisotropic behavior. In this model, the superposition of the forces from all the walls is considered.

2.4. Respect mechanism

In order to be more realistic about the collision of pedestrians, the respect mechanism introduced by Parisi et al. [2] has been implemented. In this model, a respect area is defined for each pedestrian. If this respect area is violated by any other pedestrian, the desire velocity of the pedestrian will be temporarily set to zero (only for the time steps at which this collision is happening).

The respect radius is calculated based on a respect factor, and the radius of the pedestrian:

$$d_{respect} = f_{respect} r_i$$

And the respect area is a circle with the radius equal to the respect radius and centered in a position $d_{respect}$ away from the position of the pedestrian in the desired direction.

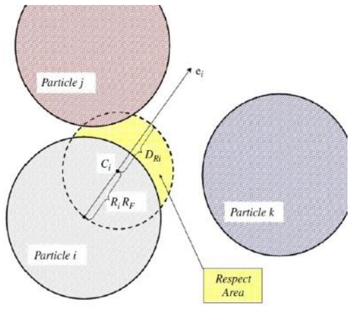

Figure 2: figure from Parisi et al[2]

2.5. Solving the differential equation

The resulting ordinary differential equation (ODE) is solved through a prediction-correction algorithm based on the velocity Verlet and Gear’s method [3,4] for each pedestrian.

The algorithm of the simulation based on this method is as follows:

```
Reading the simulation parameters
For each timestep do
    For each pedestrian do
        Calculate predictor values
        For each other pedestrian in the system do
            Calculate social force exerted by the pedestrian
        Add sum of the social forces to total force
        For each boundary element do
            Calculate forces exerted by the boundary
        Add sum of the boundary forces to the total force
        Calculate driving force
        Add the driving force to the total force
        Calculate correction values
        Calculate position, velocity and acceleration of the
        pedestrian
```

3. **Structure of the code**

The mentioned simulation process is implemented in the C++ programming language (c++ 11), with an object-oriented manner. Three classes are defined the data structures of this code:

`ParameterReader`: Can be used to read any parameters required for the simulation from the given data file.

`Pedestrian`: Represents each individual pedestrian and contains all the related properties of a pedestrian (e. g. position, velocity, acceleration, desired velocity and direction).

`Queue`: Represents a queue of pedestrians (vector of type Pedestrian) with random position and desired velocity (within the range of the desired velocity) waiting to enter the system from the desired regions. The entering time of each pedestrian is pre-defined randomly considering the flux.

As mentioned in the above algorithm, the process starts with reading and storing the parameters of the simulation from the *parameters.dat* file placed in the data folder. This work will be done through a function `parameterInitialization()`. These parameters include boundary (walls) points, flux, pedestrian radius and desired velocity and their maximum acceleration, constants of the simulation, time step size, visualization steps and duration of the simulation. Some other variables like the counters of the time loop and outputting are also defined here.

Following this step, two Queue objects queueA and queueB are created which contains pools of pedestrians waiting to enter the system during the simulation. The number of pedestrians in the queues are defined based on the flux and the duration of the simulation. These pedestrians will be added to the container of active pedestrians in the system (a vector of objects of type Pedestrian) in pre-defined random times.

Here, the main part of the program will start with a time loop. In each time step, first it will check if any pedestrian in any of the queues are ready to enter the system (to the container of active pedestrians). Then, the active pedestrians and other parameters of the simulation will be passed to the solver for calculating the position, velocity, acceleration etc. of the pedestrians.

`solver()`: The prediction-correction algorithm is implemented in this function. These steps are done by calling the following functions, sequentially:

> `predictor()`: a prediction of the position, velocity and acceleration (and its first derivative) of the pedestrian will be calculated based on the values from previous time step [3].
>
> `collisionDetector()`: this function checks if the respect area of the pedestrian is violated by any other pedestrian. If so, the collision flag will be set to true.
>
> `calcSocialForce()`: In order to calculate the social force from the other pedestrian, first the distance between the two pedestrians will be calculated and the

normal and tangential direction of this force will be defined. Then the angle between the desired direction of motion and the direction of repulsive force will be calculated to calculate the form factor. Employing these values, the socio-psychological and physical (body and sliding) forces will be calculated and summed up as the social force exerted by the other pedestrian.

`calcBoundaryForce()`: the same procedure as the social force calculation is implemented hear. To calculate the shortest distance of pedestrian and the wall, based on the position of the pedestrian, the distance to one or the other end of the wall, or the normal distance to the wall is considered.

`calcDrivingForce()`: in this function, first the collision flag will be checked to consider the respect mechanism. If it was true, the desired velocity will be set to zero for the current time step. Otherwise, the driving force will be calculated as normal.

Using the sum of all the calculated forces (driving force, sum of the forces from the other pedestrians and the walls), the true acceleration of the pedestrian will be calculated here. Then it will be checked that this value is not greater than the maximum acceleration.

The difference between the calculated acceleration and the predicted acceleration will be used as the correction factor in the correction step.

`corrector()`: using the correction factor and the predicted values the actual position, velocity, acceleration and first and second derivative of acceleration will be calculated in this function. The constants for these calculations are extracted from [3,4].

After solving the equations and calculating different properties of the pedestrians, the results will be outputted using these functions:

`VTKWriter`: this function creates the .vtk files with the given frequency for visualization (e.g through paraview).

`CSVWriter`: creates one .csv file for postprocessing, storing the information about the pedestrian at each time step.

3.1. How to use the code

The code can be easily compiled with the provide make file, using the g++ compiler.

For compiling: `make`, For running: `bin/pedSim.exe`

The parameter of the simulation can easily be changed in the *parameter.dat* file and recompilation is not required after parameter changes. The functionality of the code has been tested performing some cases with single and two pedestrians and the corresponding videos are provided in the test folder.

4. Results

Table 1 represents parameters of the system gathering from literatures [1, 2, 4, 5] or with calibration (e. g. relaxation time of 0.5 was too harsh for the higher flow rate, after testing different values, a value of 0.4 has been used in the simulation).

Time step size	0.1
Relaxation time	0.4
Interaction strength (pedestrians)	2.1
Interaction range (pedestrians)	0.3
Interaction strength (walls)	5.1
Interaction range (walls)	0.2
Lambda	0.2
k	2
κ	4.8
Respect factor	0.5

Table 1: Parameters of simulation

Simulations are done with the different pedestrian flux at each inlet. Figure 3 represents different visualization of pedestrian motions for increasing flux. The animation of the motion of pedestrians can be found in the post-processing directory. As expected, the motion of pedestrians is undisturbed at lower fluxes and they walk through a nearly straight line, except when they are close to the wall. The interactions between the pedestrian at the intersection are also low and their velocity is nearly constant, except during the acceleration time at the beginning. As the flux increases, the pedestrians interact more with each other at the intersection and their walking line skews to right (for A-B pedestrians) or upward (for C-D). At very high fluxes, some of the pedestrians went to the other corridor and got lost there and could not find their way toward the destination, so stopped near the wall of the other corridor. Furthermore, the density of the pedestrians at both corridors after the intersection, is higher in the skewed side.

Figure 4 shows the mean velocity vs. density of the pedestrians at the intersection for the highest flux, which could reproduce the general trend of the fundamental diagram. However, it decays faster than the ones showed in literature (e.g. [6]), which is clearly because of the existence of the intersection in our model.

The mean walking time of the pedestrians and outflux of the gates B and D are also demonstrated in Figs. 5 and 6. By increasing the interactions between the pedestrians in higher flow rates, the pedestrians require more time to take the path. the mean outflux also seems to be exhausted after a critical influx. The pedestrians interact more and get stuck at the intersection.

Project: Simulator of the pedestrians' dynamics
Writer: Hossein Sadeghi
Date: 26.09.2020

Flux (p/s) **A: Pedestrians positin** **B: Walking lines (A-B)** **C: Walking lines (C-D)** **D: Force heatmap** **E: Velocity heatmap**

0.2

0.5

1

1.5

2

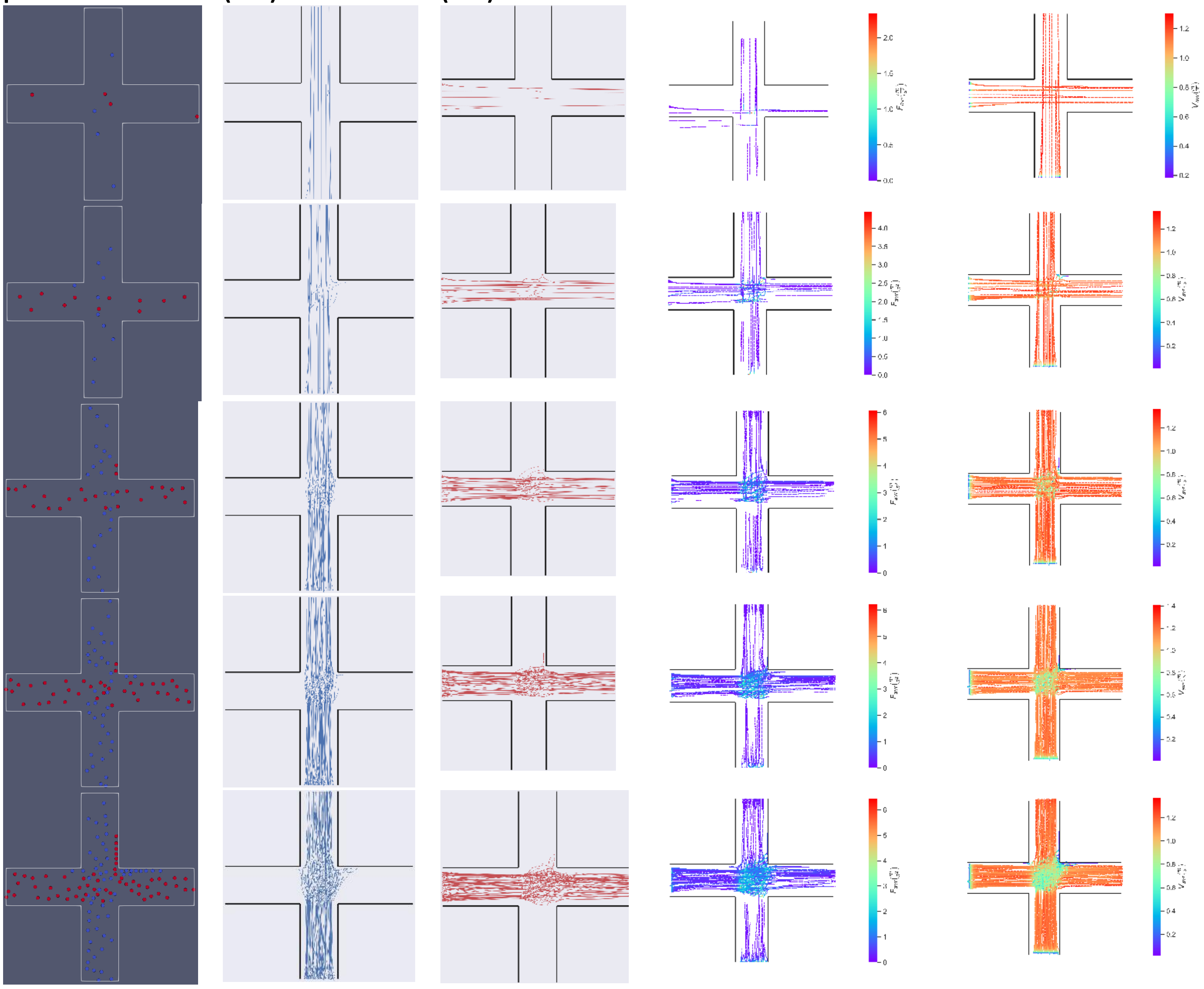

Figure 3: A) Visualization of pedestrian position at t=55s, average pathlines for B) A-B and C) C-D corridors, heatmaps of the average D) forces (boundary+social) and E) velocity

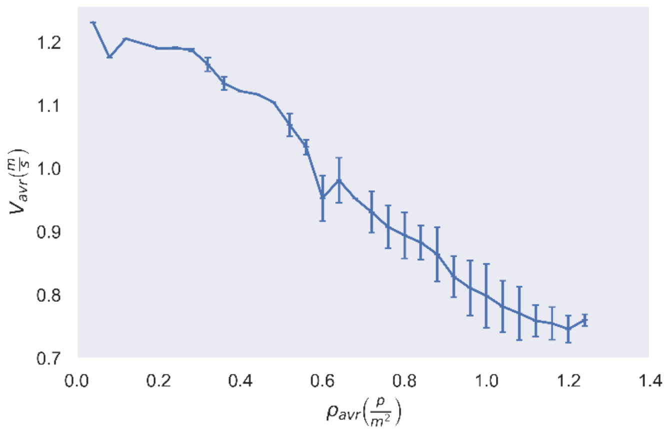

Figure 4: Mean velocity vs. density of the pedestrians at the intersection for flux=2 p/s. The error-bars show the standard deviations.

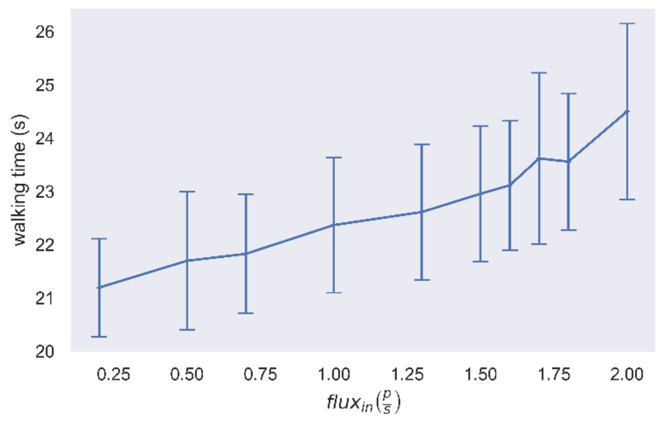

Figure 5: mean walking time of the pedestrians which successfully reach the outlet gates. for different fluxes.

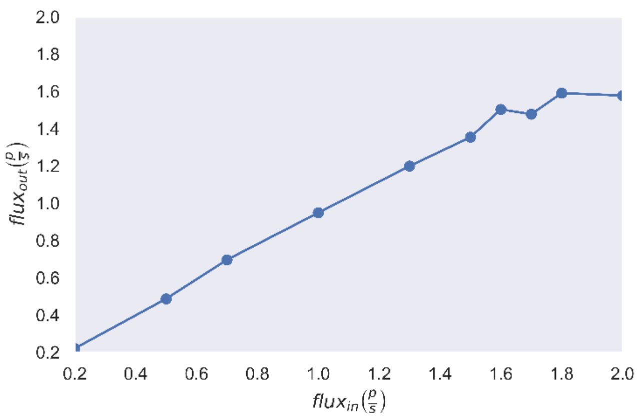

Figure 6: Mean flux out of the corridors through gates B and D (average of them) vs. the entering flux.

5. Outlook

In this model, the attractive forces between the pedestrians like their joining behavior and also toward the other attractions is neglected. The model can be improved by adding these forces to the model [4].

Further improvement can be done by considering the unsystematic random behavior of the pedestrians which can be added to the total force with a Gaussian distribution [2].

Although, a respect mechanism has been used in this mode, the interaction of pedestrians with an intersecting direction of motion is still problematic. Now, the pedestrian collides each other, then try to change their way. However, this clearly is not real human behavior. In order to handle such interaction, a conflict avoidance mechanism proposed by Li et al. [7] can be added to the model, which predict potential conflict by considering direction of the velocity vectors of pedestrians, the time they reach the intersection and a safety interval.

Moreover, the graphs of higher flow rates show higher density of pedestrians in the skewing side after intersection. Some pedestrian also got lost in the other corridor. By adding a recoiling force and calibrating the required constants, we may get a more logical behavior in such situations.

Finally, the performance of the code can be improved through some steps. In the current model at each time step we have a nested loop to search the pedestrians. This causes a complexity of n^2 for the simulations. However, it is not necessary to search all the pedestrians far away from each other. The exponential term in the forces decays very fast after some distance. For the current simulations with several hundred pedestrians this is not a problem, but when thousands of pedestrians exist in the system this causes very high computational costs. Therefore, two approaches may be followed here to improve this behavior: the Verlet algorithm which search only a definite radius around the pedestrians and the linked-cell algorithm with a neighboring table [4]. This way only the pedestrian within the neighboring cells in each time step have to be considered.

6. References

[1] Helbing, D., & Molnar, P. (1995). Social force model for pedestrian dynamics. Physical review E, 51(5), 4282.

[2] Parisi, D. R., Gilman, M., & Moldovan, H. (2009). A modification of the social force model can reproduce experimental data of pedestrian flows in normal conditions. Physica A: Statistical Mechanics and its Applications, 388(17), 3600-3608.

[3] Allen, M. P. (2004). Introduction to molecular dynamics simulation. Computational soft matter: from synthetic polymers to proteins, 23(1), 1-28.

[4] Apel, M. (2004). Simulation of pedestrian flows based on the social force model using the verlet link cell algorithm. Poznan University of Technology, 79.

[5] Seer, S., Rudloff, C., Matyus, T., & Brändle, N. (2014). Validating social force based models with comprehensive real world motion data. Transportation Research Procedia, 2, 724-732.

[6] Weidmann, U. (1993). Transporttechnik der fußgänger: transporttechnische eigenschaften des fußgängerverkehrs, literaturauswertung. IVT Schriftenreihe, 90.