ETHlogo

**Lecture with Computer Exercises:**

**Modelling and Simulating Social Systems with MATLAB**

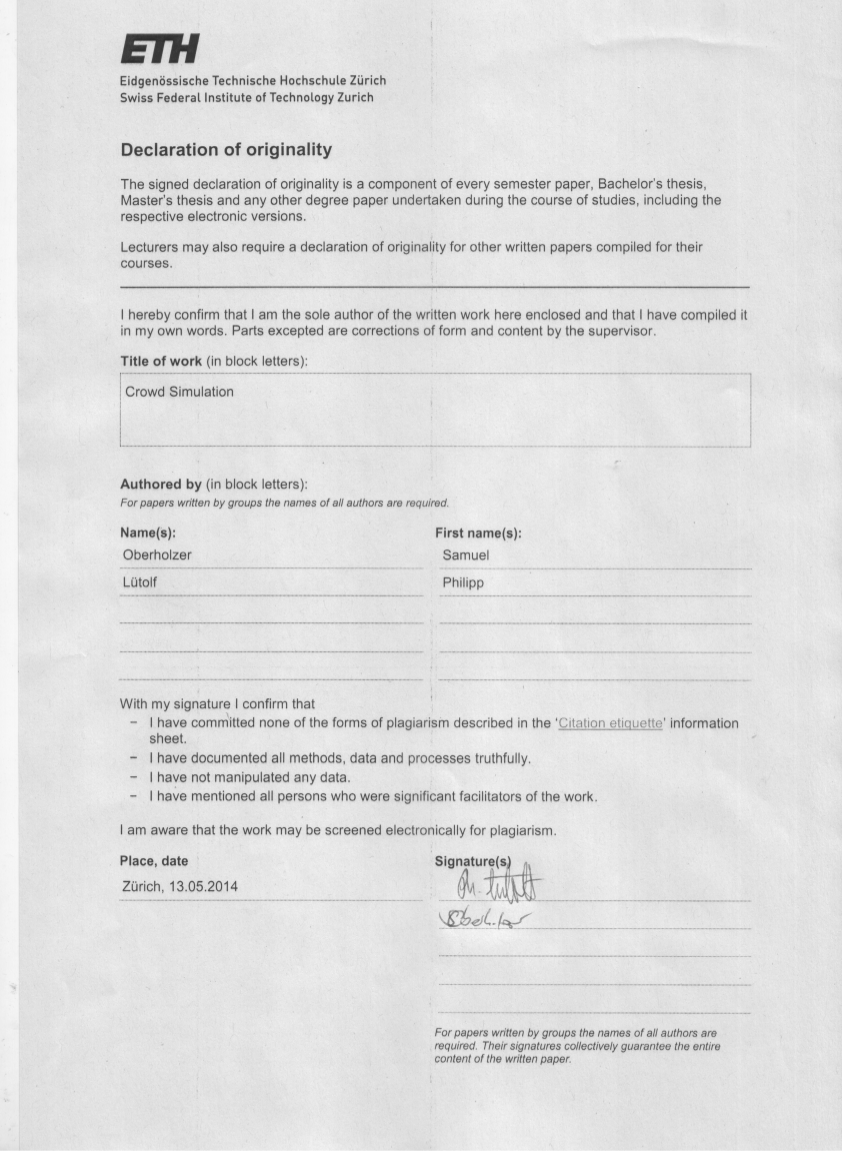
Project Report

|  |
| --- |
| Crowd Simulation  … |

Samuel Oberholzer & Philipp Lütolf

Zürich

11.05.2014



**Agreement for free-download**

We hereby agree to make our source code of this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

|  |  |
| --- | --- |
| Samuel Oberholzer | Philipp Lütolf |

**Table of Content**

[1 Abstract 5](#_Toc387773751)

[2 Individual contributions 5](#_Toc387773752)

[3 Introduction and Motivations 6](#_Toc387773753)

[4 Description of the Model 7](#_Toc387773754)

[4.1 Social force model 7](#_Toc387773755)

[4.2 Acceleration force 7](#_Toc387773756)

[4.3 Pedestrian Interactions 8](#_Toc387773757)

[4.4 Boundary Interactions 8](#_Toc387773758)

[5 Implementation 10](#_Toc387773759)

[5.1 Initialization and Simulation overview (the testModel.m file) 10](#_Toc387773760)

[5.2 Initialization 10](#_Toc387773761)

[5.2.1 Walls and Waypoints 11](#_Toc387773762)

[5.2.2 Agents 11](#_Toc387773763)

[5.2.3 The “cross”-situation 12](#_Toc387773764)

[5.2.4 The “curve”-situation 12](#_Toc387773765)

[5.3 Simulation 13](#_Toc387773766)

[5.3.1 Setting the next destination 13](#_Toc387773767)

[5.3.2 The acceleration force 14](#_Toc387773768)

[5.3.3 The wall force 14](#_Toc387773769)

[5.3.4 The pedestrian force 14](#_Toc387773770)

[5.3.5 Position update and saving data 14](#_Toc387773771)

[5.4 Plotting the data and making a video (simulate.m) 15](#_Toc387773772)

[5.5 Overview of the result (maxPeopleOnSquare.m) 15](#_Toc387773773)

[5.6 Model constants 15](#_Toc387773774)

[6 Simulation Results and Discussion 17](#_Toc387773775)

[6.1 Simulation without obstacles 17](#_Toc387773776)

[6.2 Simulation with obstacles 18](#_Toc387773777)

[7 Summary and Outlook 18](#_Toc387773778)

[8 References 18](#_Toc387773779)

[9 Appendix 18](#_Toc387773780)

# Abstract

The goal of the simulation is to model big crowds in different locations and identify the dangerous spots. Our model is programmed in Matlab, is agent – based and in continuous space. The physics are based on the ‘social force model’ from Helbing. To identify the dangerous spots, the density of the people is computed and illustrated in the simulation. In this report two different locations are considered, a crossroads and a ninety-degree curve which are simulated with and without obstacles to identify the differences.

The results are then compared to real life experiences and to the results from similar projects.

# Individual contributions

The work on this project was shared evenly between both authors because we always worked together.

# Introduction and Motivations

Big events with a lot of people bear a great risk of mass panic. There are numerous events from the past where people got injured or died because of the forces acting on pedestrians in dense crowds. The most recent crowd disaster was the love parade in Duisburg 2010, where 21 people died and over 500 people were injured.

One critical factor of such tragedies is the location. Obstacles, corners, intersections and other bottlenecks where the crowd is compressed are the most critical spots. The main questions which we are going to answer are the following:

Where are the most critical spots for people in a crowd in a given environment?

What are the possibilities to reduce the risk to individual people by rearranging the environment?

To answer these questions we decided to use the “social force model” by D. Helbing [1]. This model is able to reproduce many real life situations and is explained in section 4. We have decided to implement this model in Matlab as an agent-based model in continuous space. To model different arrangements we chose to model two different situations, each one with and without obstacles. One situation is a crossroads and the other situation is a ninety-degree curve. Explanations concerning the implementation are given in section 5.

Our results are then compared to real life experiences and the results of other projects from previous semester (section 6).

# Description of the Model

## Social force model

The model is based on the ‘social force model’ developed by D. Helbing and P. Molnar [1] [2]. It describes the “social forces” acting on a pedestrian in a crowd. Each pedestrian in a crowd can be represented with a point in space, the speed of the pedestrians can be described by the equation

and the acceleration by

.

D. Helbing and P. Molnar added a fluctuation term which include random variations of the behaviour.

One term of describes the driving force which accelerates the pedestrian towards the desired velocity , another term consists of the repulsion force from other pedestrians and obstacles and the last force outlines attractive effects .

## Acceleration force

The driving force accelerates the pedestrian towards the destination which results in the desired direction

Deviations of the actual velocity from the desired velocity occurring from obstacles or other pedestrians are corrected within the “relaxation time” .

The desired speed is defined as

Where is the initial velocity and the maximum desired velocity.

The parameter

Characterizes the nervousness of the pedestrian to reach their destination, where describes the average speed of the pedestrian.

## Pedestrian Interactions

The term describes the repulsive force from another pedestrian . This force is defined as

Where denotes the respective interaction strength and the range of the repulsive interaction. The parameter is the sum of the radii of both pedestrians, is the distance between the centres of mass and is the normalised vector pointing from to .

The last parameter accounts for the directionally dependent behaviour of pedestrians. In the context of crowds the factor gives the pedestrians within sight greater influence than those out of sight.

Where the parameter characterizes the directionally dependent behaviour with

and

## Boundary Interactions

The force acting on pedestrians from boundaries is almost identical to the repulsion force from other pedestrians. The only difference is that the directionally dependent behaviour can be neglected.

Where is the normal vector pointing from the boundary to the pedestrian and the distance between the boundary and the pedestrian.

# Implementation

The social force model is implemented in Matlab as an agent – based model in continuous space. The implementation consists of three parts. First, the testModel.m file where all the computation is done and the data is saved. Secondly, a simulate function, that plots the saved data and makes a video out of it. And third, the maxPeopleOnSquare function, which visualizes the simulation data for analyzing it.

## Initialization and Simulation overview (the testModel.m file)

The implementation of the core file (testModel.m) roughly follows the Pseudocode shown below:

—Initialization—

Set walls and waypoints

Set start positions of agents

—Simulation—

FOR each frame of the simulation

FOR every pedestrian in the system do

Set up next destination and calculate desired direction

Calculate acceleration force and influence on velocity

FOR each wall element do

Check distance to element

Calculate wall force of closest element and influence on velocity

END

FOR every other agent of the matrix do

Calculate pedestrian force and influence on velocity

END

Update position according to calculated velocity

save agent in a updated matrix

END

set matrix to updated matrix

save data for plotting

END

## Initialization

The implementation allows to specify and run the simulation on different maps, only depending on the initialization of the walls, waypoints and agents. Two situations were specified to investigate our research questions.

### Walls and Waypoints

Walls are specified by a start- and endpoint. So every wall needs to store x and y position for two points, means four values per wall. Obstacles can be placed into the scene as well by placing them as walls.

Because it’s unlikely, that all agents head to a single point in space, a new concept of waypoints is introduced. Here, waypoints are more like „waylines“ and stored in the same format as walls. It’s a line that fixes the next destination of an agent. The shortest path to this line is used to calculate the desired direction of an agent. If an agent reaches the last waypoint, he respawns at his starting area.

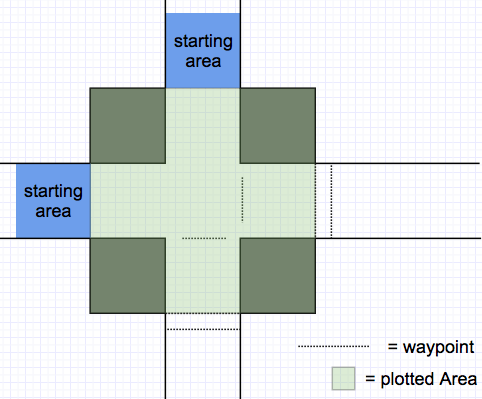
### Agents

There is one matrix containing all the agents of the simulation. Every agent consists of eight different values. These are the agent’s position and its current velocity, its desired velocity which is Gaussian-distributed at and its type that determines to which group of agents the agent belongs and therefore where it’s starting area and destination is. Further, which waypoint the agent currently is aiming for, and what its average speed is.

The positions and waypoints for the initialization depend on the situation. Here, the following two situations were specified:



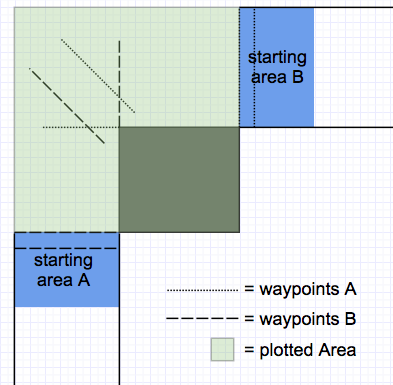
### The “cross”-situation



The walls are defined as seen in the picture above. Agents are initialized randomly distributed in a 5x5 area on the left side and on the top (outside the plotted area). Their goal is to reach the other side of the cross-way. To achieve this, the way points were set as seen in the picture.

The crowd flow in this situation is investigated with and without the obstacles seen in the picture and the outcome is compared.

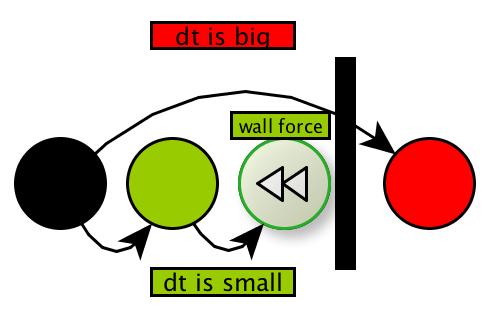
### The “curve”-situation



In this situation, the goal of the agents is to follow the street and make it to the other end. One half starts at the bottom-left end and the other half at the top-right. Here also obstacles were added to compare the crowd flow with obstacles to the one without. The waypoints are set as seen in the picture to overcome the obstacles.

## Simulation

In order to realistically simulate movement of agents in continuous space, they need to be updated a lot. Time between these updates was chosen at 0.05s, which corresponds to 20 frames per second. If the time between the updates is chosen too high, the steps of the agents are larger and important factors influencing their movement could be skipped. For example if an agent moves towards a wall and the update occurs shortly before the agent is pushed back by the wall force, the agent could be past the wall in the next update.



In every step, all the agents have to be updated. These updates depend on the current waypoint they are aiming for, the walls, and the positions of other agents. So for every agent is done the following things:

### Setting the next destination

The distance from the agent to the next waypoint is computed with *vectorFromWall*. Is the distance small enough, the next waypoint is set as new destination.

If it was the last waypoint, the agent has reached its destination and is re-initialized at the starting area. Like this, the flow of incoming agents can be maintained without having to create new ones and therefore performance is tuned. The desired direction is computed again using *vectorFromWall* so the acceleration force can be computed.

The function *vectorFromWall* takes a waypoint and the agent’s position and gives back the normalized vector pointing to this waypoint and the distance to it.

### The acceleration force

The acceleration force is computed using *accelerationF*. Then the force is added to the current velocity of the agent.

The function *accelerationF* takes the desired direction, current velocity, average speed and the desired velocity and returns the calculated acceleration force according to section 4.2.

### The wall force

Next, the nearest wall is searched using *vectorFromWall* again, then the wall force is computed using *wallF*. The wall force is added to the velocity afterwards

The function *wallF* takes the distance to a wall and the normalized vector pointing to it. According to section 4.4 the wall force is computed.

### The pedestrian force

Every other pedestrian has a little influence on the agent. Therefore, the pedestrian force resulting from every other agent is computed using *pedestrianF* and added to the velocity of the agent. Note that the loop over all the agents uses the matrix from the last step, so updates on all agents are made simultaneously.

The function *pedestrianF* computes the pedestrian force using the positions of two agents as arguments according to section 4.3.

### Position update and saving data

After all the forces contributing to the new velocity have been computed, the position of the agent is updated. Additionally, the matrix containing information about how many people are in a square meter is updated.

After all the agents have been updated, the whole matrix is updated simultaneously. The positions of the agents are stored separately so that the simulation run can be plotted later on.

## Plotting the data and making a video (simulate.m)

*function simulate(filename,mode,savevideo)*

In this function, the saved data from the file ‚filename’ is visualized. The x positions of all the agents are plotted against their y position. Combined with the plot of the walls matrix this results in the visual simulation of the situation. If ,mode‘ is passed a 1, the whole plot is provided with a background picture. This background picture shows how many people are standing in each square meter. The whole plot is saved to a video file called ,savevideo’.

## Overview of the result (maxPeopleOnSquare.m)

*function maxPeopleOnSquare(inputfile)*

This function creates a figure with two subplots to summarize the data of the plot. Input data is loaded from the file ,inputfile’. The first plot visualizes what the maximum number of people standing in a square meter was at each frame of the video. The second plot shows for each square meter of the map, what the maximum number of people on it was throughout the whole video.

## Model constants

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Parameter** | **Section** | **Value** |
| Initial desired velocity |  | 5.2 |  |
| Relaxation time |  | 5.2 |  |
| Maximum speed |  | 5.2 |  |
| Territorial sphere pedestrian interaction strength |  | 5.3 |  |
| Territorial sphere pedestrian interaction range |  | 5.3 |  |
| Anisotropic character |  | 5.3 | 0.75 |
| Physical pedestrian interaction strength |  | 5.3 |  |
| Physical pedestrian interaction range |  | 5.3 |  |
| Boundary interaction strength |  | 5.4 |  |
| Boundary interaction range |  | 5.4 |  |
| Radius of pedestrian |  | 5.4 |  |

The model constants were chosen as proposed by Helbing [2].

# Simulation Results and Discussion

As stated before, two different situations were considered. The situation with a ninety degree corner (section 5.2.4) and the “cross” situation (section 5.2.3). The results from our simulations without obstacles are presented in section 6.1, where the main goal was to identify the critical spots. Further simulations included bottlenecks and obstacles which are discussed in section 6.2. In these simulations we examined what influence the obstacles and bottlenecks had and if they reduced or increased the safety of the pedestrians.

For our simulations the following parameters had to be set:

* Number of pedestrians
* Time frame
* Obstacles
* Situation

## Simulation without obstacles

For the simulation without obstacles we chose the number of pedestrians to be 200 and the time frame was set to 30 seconds.

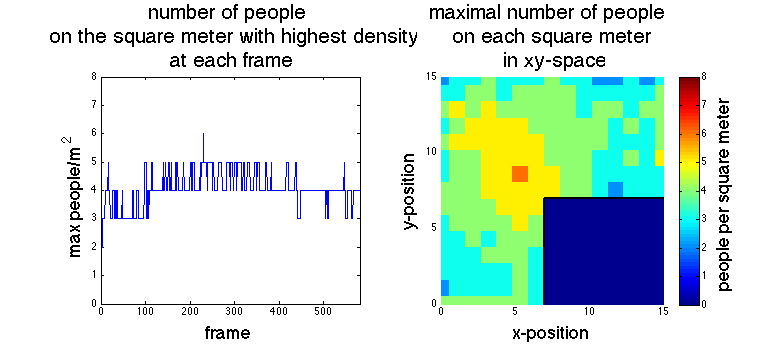


Figure 1:

## Simulation with obstacles

# Summary and Outlook

The chosen constants make the model work best in denser crowds., One could adjust the computation of the acceleration force so the simulation becomes more realistiv for situations with fewer people. One could also implement attraction effects as mentioned by Helbing [2].

Further improvements of our implementation could be made. For better performance, it would be best if pedestrian force was only computed amongst agents that are near to each other. For this purpose one could store agents in a quadtree, this would tune performance a lot.

More important though is the fact, that agents still influence each other a lot, if they are near each other but separated by a wall. This has an influence on the quality of the simulation. Therefore, pedestrian force computation should consider whether between agents is a wall.For analyzing the density of the crowd, a different method could be applied. Instead of discretizing the space into square meter “buckets” that count people on them, one could

Take a finer mesh grid and calculate for every point how many people are in the square meter around that point.

All in all, there are no significant new insights gained. However, the implementation comes up with similar results as Helbing in [2] and backs up personal experiences gained in a curve like situation. Further, a reusable implementation is presented to which a lot of different situations could be applied.

# References

|  |  |
| --- | --- |
| [1] | P. M. D. Helbing, "Social force model for pedestrian dynamics," *Physical Review E,* vol. 51, no. 5, 1995. |
| [2] | L. B. A. J. T. W. D. Helbing, “Self-Organized Pedestrian Crowd Dynamics: Experiments, Simulations, and Desing Solutions,” *Transportation Science,* vol. 39, no. 1, pp. 1-24, 2005. |

# Appendix