Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture with Computer Exercises: Modelling and Simulating Social Systems with MATLAB

Project Report

Newspaper Boxes: How do they Influence the Pedestrian Flow?

Dario Kneubühler, Roman Müller,
Anja Sutter, Ueli Wechsler

Zürich
14th December 2012

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.

Dario Kneubühler

Roman Müller

Anja Sutter

Ueli Wechsler

Abstract

Train stations in Switzerland are frequented by loads of people every day. Since about 10 years, free newspapers are disposed in boxes. Our aim was to find out where the best placement of a newspaper box in the main crossing area in Lucerne train station is. We used the Social Force Model to simulate the pedestrian behaviour and run our model with six different paper box positions. The measurement of the pass-through times of the agent revealed that a box placement at an edge provokes more crowding and more waiting times around the box than a placement in the middle of the free space. Also, the orientation of the box can have a strong influence on the pedestrian flow. However, in order to justify our results, a detailed evaluation of the chosen parameters would have to be executed.

In order to enhance the readability of this text, we used ‘he’ as a personal pronoun. However, male and female persons are meant equally.

Table of Contents

1. Individual contributions

In the beginning, ideas for our projects and on how to implement the simulation were discussed together. The code was written by Dario Kneubühler, Roman Müller and Ueli Wechsler. All group members, led by Ueli Wechsler, executed simulations and analysis. Also all group members, led by Anja Sutter, wrote the report.

2. Introduction and Motivations

Every day many people pass Lucerne train station (Figure 1) and several simulation programmes were implemented to study the interaction between geometry of the building and the passenger's flow (Emch+Berger AG Bern). With the emergence of free newspapers (like "20 Minuten", "Blick am Abend") about 10 years ago, this problem got a new dimension: If a commuter is heading towards the newspapers box, he often has to cross the other people's way.

Figure 1: Lucerne train station at 17.30h.

In rush hours this can strongly influence the general passenger flow when entering or leaving the platform and the station. How much can such boxes slow down pedestrians? Where are their best locations, such that all pedestrians (including those who are picking up a newspaper) can pass the train station as fluently as possible?

2.1. Fundamental Question

Based on these reflections, we phrased our research question: How do different newspaper box placements influence the time, in which the pedestrians pass a certain route?

2.2. Research Methods

We used the layout of Lucerne train station, of which we simulated one important crossing area. In order to find the ideal location for a newspaper box we chose six different positions and ran the simulation four times for every position. For the different paper box placements, we evaluated the time a person needs to get from one entry line to another exit line in the simulation. Based on this we chose the situation with the highest median crossing time and the one with the lowest median crossing time and discussed the two situations in detail.

The final output let us conclude how strongly an unfavourable placement of newspaper boxes influences the pedestrian flow.

2.3. Simulation

The simulated crossing area has five entries. We decided to simulate a situation when a train is entering the station. In the beginning, nobody is coming from the platform. After a while, train passengers enter the simulated crossing area with an increased probability to pick up a paper. After a defined number of passengers, the train is empty and again, nobody is coming from the platform. During all this time, other agents are crossing the area from the other entries to the platform or to one of the other exits.

2.4. Expected Results

We expected to get a relevant difference between the placement options. Furthermore, we expected that a box positioned in the middle of a corridor (compared to a box standing at the edge) will slow down the passenger's flow if the number of paper-takers is low, but might be preferred if a high ratio of passengers is taking a newspaper, because a smaller part of the pedestrian flow has to be crossed to pick up a newspaper. Also, people picking up a newspaper can form a crowd, which can lead to a bottleneck situation if the percentage of paper-takers is high.

3. Description of the Model

3.1. Social Force Model

Our goal was to implement the pedestrian simulation model that is best suited for our research purpose. We decided to use the Social Force Model since it is easy to adapt and was used with success for similar projects (Heer & Bühler 2011; Vifian, Roggo & Aebli 2011).

The model describes pedestrians as particles that are driven by forces. Our implementation is based on "Social force model for pedestrian dynamics" by Helbing & Molnar (1995).

3.1.1. Destination Force

Each pedestrian aims to reach a certain destination. They will most likely choose the shortest possible path, which usually is of the shape of a polygon. This polygon is given by

$$\vec{e}_i(t) = \frac{\vec{r}_a^k - \vec{r}_a(t)}{||\vec{r}_a^k - \vec{r}_a(t)||}$$

where $\vec{r}_a^k$ is the next edge of the polygon and $\vec{r}_a(t)$ the current position of the agent.

Taking into account the pedestrians current and desired speed results in the force F' that points in the pedestrians desired direction.

In order to solve this problem we used a version of the Fast Marching Algorithm. It calculates the gradient field of the shortest path for all 6 possible desired destinations (five exits and the paper box).

$$\vec{F}_i' = \frac{1}{\tau}(v_{desired} - v_{actual})$$

Also the direction of each agent is changed by a Gaussian distributed random angle with a tendency to prefer the right hand side (since this is the preferred direction in continental Europe). This fluctuation is used to represent a random variation of the behaviour of each pedestrian, for the cases in which two or more behaviour options are equivalent.

$$alpha = R * arc + tend$$

Alpha is a random angle calculated form a Gaussian distributed random number (R), a defined angle (arc) and a tendency (tend).

3.1.2. Pedestrian Force

This is the force that controls the interactions between pedestrians. Pedestrians like to have a zone of privacy around them. The closer they get to a stranger the more uncomfortable they feel. Also actions that happen in front of the pedestrians will affect them more than actions that occur behind them. This is taken into account by a sphere of privacy around each pedestrian, which has the form of an ellipse pointing into the direction of his velocity. F'' is the repulsive force of a pedestrian, where v is the potential field surrounding each agent and $D_{i,j}$ is the distance between the pedestrians i and j.

$$\vec{F}_i'' = -\sum \nabla v(D_{i,j})$$

3.1.3. Wall Force

Agents are repulsed by walls and other obstacles that might be in their way. Therefore every wall is surrounded by a potential field that leads to a force F''', which can be described by

$$\vec{F}_i''' = -\nabla U_B(R_{i,B})$$

where U_B is the potential of the wall and $R_{i,B}$ is the distance between the wall and the pedestrian. The repulsive force increases exponentially the closer the agent gets to the wall, thereby avoiding that agents bump into walls or get stuck.

3.1.4. Total Force

Superpositioning the three forces plus an additional fluctuation term results in the total Force given by

$$\vec{F}_i^{total} = \vec{F}_i' + \vec{F}_i'' + \vec{F}_i'''$$

3.2. Path Definition

3.2.1. Polygon

In order to define the agents' path to their goal, a first approach was made by implementing our own code. The general idea was to look at the line between the agent and his final destination. If there are any walls between them the destination was changed to a secondary destination being in the corner of the obstacle, which leads to the shortest path to the final destination. These steps were repeated for any obstacles that might be in the agent's way.

After some time we realized that implementing our own idea for this problem would take too much time and we decided to use the Fast Marching Algorithm instead, which was used with success in similar projects (Heer & Bühler 2011; Vifian, Roggo & Aebli 2011).

3.2.2. Fast Marching Algorithm

The Fast Marching Algorithm is based on the Eikonal equation. It is a numerical method of solving the shortest path problem quickly (Malladi, Sethian 1996; Peyre 2008). In our project it was used to calculate the gradient fields for the destination forces.

It creates a gradient that shows the shortest distance to every point on the map from a given starting point. If we assume constant velocity this is equal to travel time.

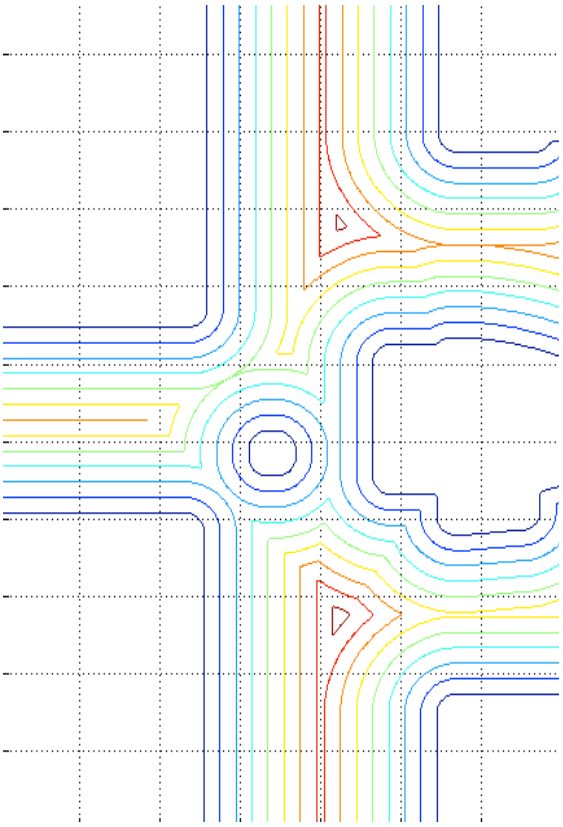

Figure 2: Distance Map of the walls. The grid distance is 50x50 pixels, or 3.3x3.3m

4. Implementation

Our implementation is divided into three parts:

- First, there is the function *runfast.m*. It is responsible for the calculation of the gradientfields, which remain unaltered during the entire simulation and also need a vast amount of time to be executed. Therefore, in order to improve the performance, these implementations are in a separate part of the programme.
- The main point is the function *simu.m*, which not only runs the full simulation but also saves the data. It is mostly an adaption of the Social Force Model (see chapter 3.1).
- Finally, there is a function *finalanal.m*, in which all results from the simulation are combined and interpreted.

Schematic overview

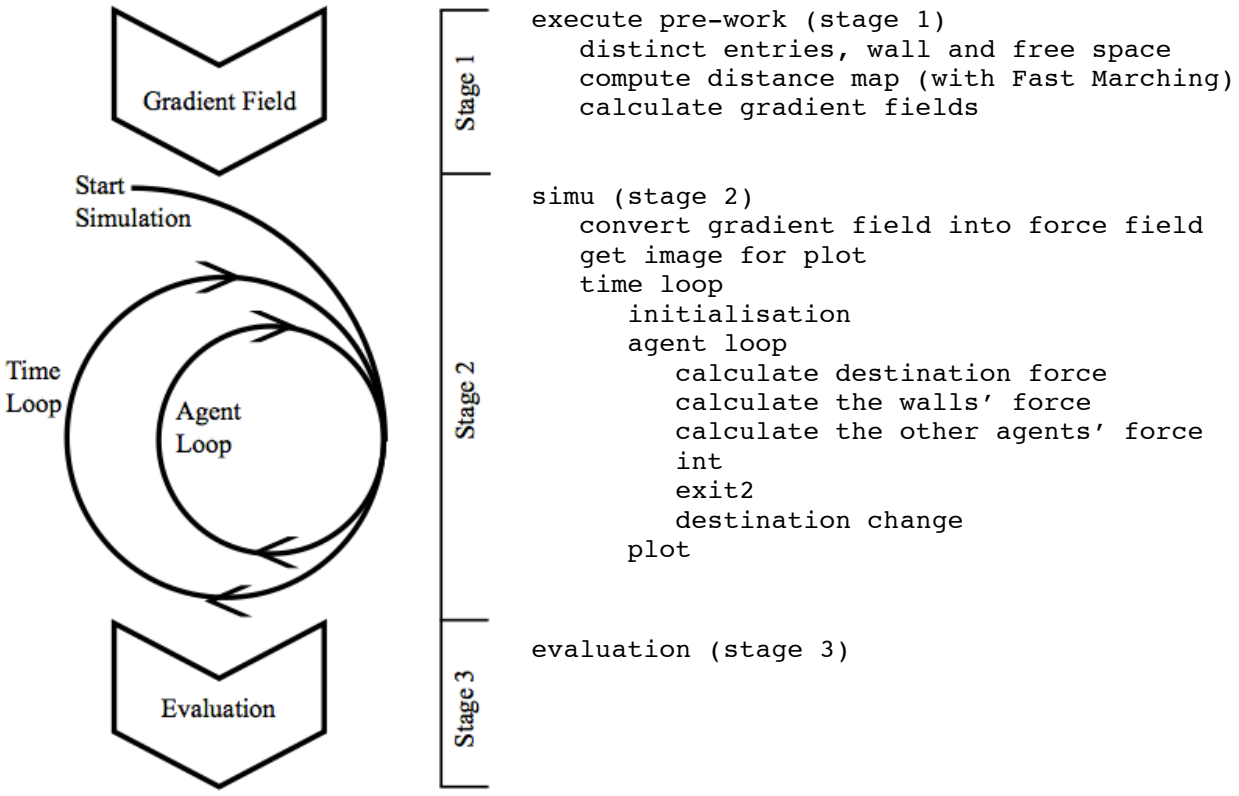

Figure 3: Schematic overview of the implementation.

4.1. Spadework

Before running the simulation, *runfast.m* executes some pre-work.

The area for the simulation is a crossing area in Lucerne train station. We made a colour-coded outline map (Figure 4) with the paper-box positioned diversely. Because of the small number of different situations we decided to make the box changes graphically rather than by a function, which could process the inputs x-label, y-label and position-angle.

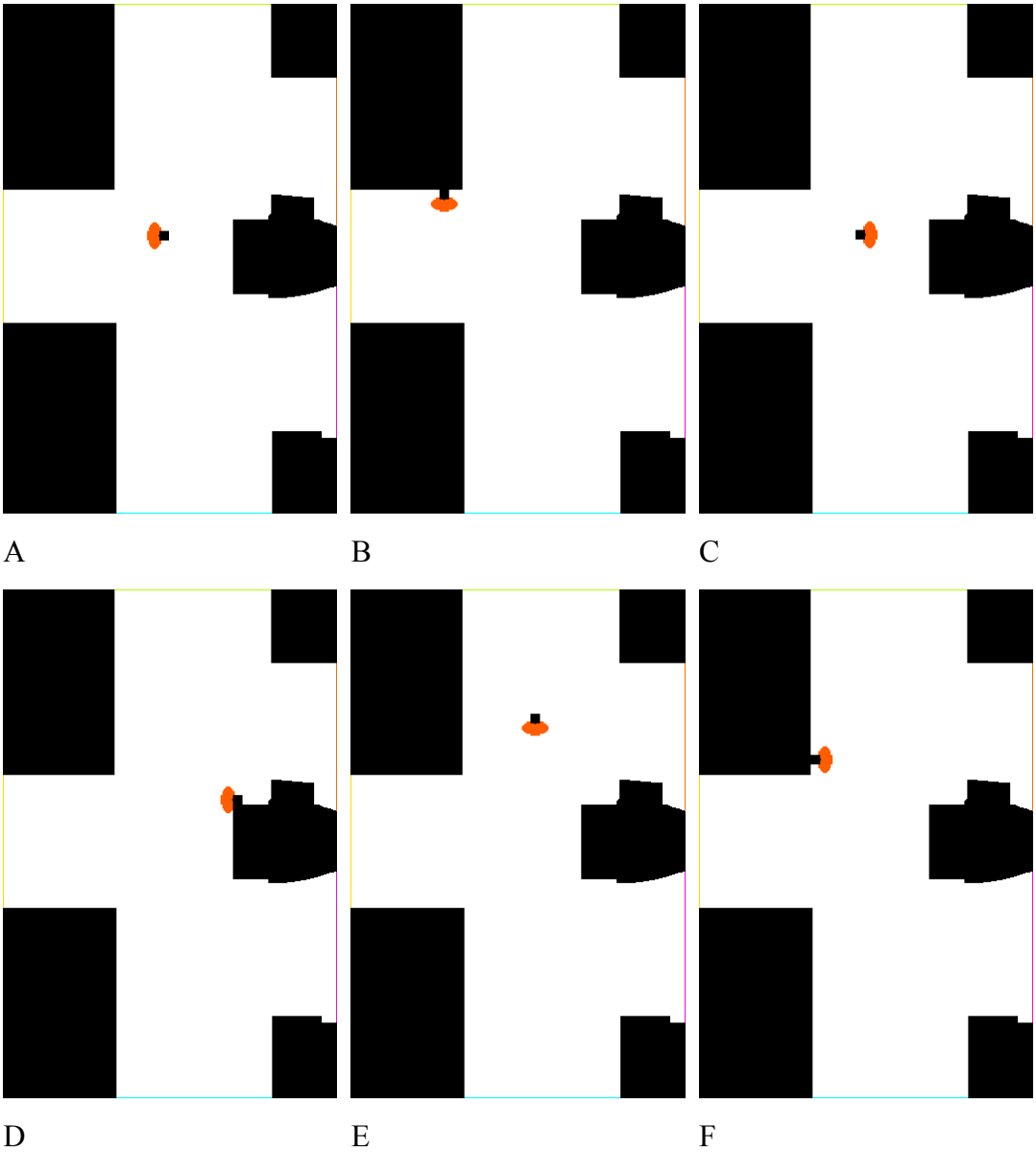

Figure 4: The colour-coded maps with the six chosen paper box positions. The orange area next to the paper box is the paper box area. The coloured lines at the entries are possible starting and ending points of the agents.

This color-coded map is turned into a single matrix with distinct entries for every attraction-region, the walls and the normal space by the function *getmap.m*.

The wall and space inputs are further used in the function *grad_walls.m* to create a gradient field, which solely displays the interaction between walls and agents as a force field.

For the actual attraction gradient field the last part of this spadework function is responsible. For every of the 6 attraction centres (the 5 exits and the box) it produces a specific distance map with the Fast Marching Algorithm (*computeGradientField.m*). With a slightly altered version of the *gradient_special.m* function from Heer & Bühler (2011) the distance maps are converted into a normalised gradient field, which is used to assign the right direction vector to every agent in a later stage.

All these direction fields packed together in a R^3-matrix for the 6 possible positions for the box plus the force field of the wall is saved at the end of the execution. These data are the basis of the simulation. Thanks to the separation of this pre-work from the programme the simulation is much faster.

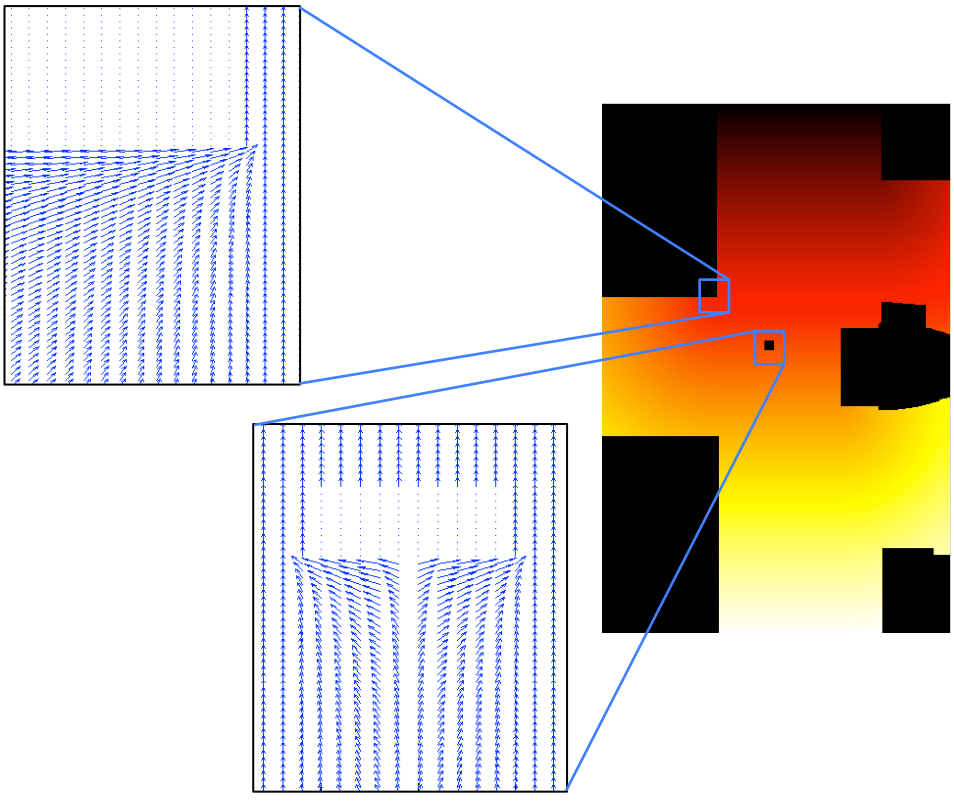

Figure 5: A distance map (right) with the entry in the east; the brighter the colour, the further away is the point. Black is for the buildings, walls and the paper box. Two detailed views of the vector field with entry in the west (left).

4.2. Simulation

The actual simulation is run by *simu.m*. Before the first time loop starts, the gradient vector field from *runfast.m* is translated into a force field and the map of Lucerne station is plotted (*getimage.m*). After this, the time loop is started. Within the time loop, agents are produced and the corresponding parameters are put into matrix *A* (with *initialisation.m*):

Start time	The time at which an agent entered the simulated area is saved in order to calculate the passing time in the end.
Number of the start position	The code number of the exit is defined with certain probabilities defined in the file *parameters.m*. Agents from the train are added after a defined time, in a defined number per second, with a defined number in total. After all the agents coming from the train have entered the simulated area, no more agents are entering through the southern entry (goal #2).
Start position	The exact starting position in the randomly defined entry is defined randomly as a vector. In the further time loops, this position is continuously updated corresponding to the current standing position of the agent.
Velocity	The velocity is defined as a vector. In the further time loops, this position is continuously updated. The probabilities are defined in the file *parameters.m*.
Paper-taker yes or no	Every agent gets a random characterisation of whether he will pick up a paper or not. This is not changed any more. The probability depends on where the agent is coming from (again defined in *parameters.m*).
Current goal	We defined a code number for every possible goal (Figure 6). If the agent is a paper-taker, his current goal is #6 for paper box, if he is not a paper-taker his current goal (#1-#5) is equal to his final goal. The probabilities for definition of the goals are set in *parameters.m*.
Final goal	The final exit is already defined in the beginning; it cannot be changed any more and is different from the entry. The corresponding probabilities are again set in *parameters.m*.

Time inside paper box area	The time every agent who picks up a newspaper spends inside the paper box area is set to 2.5 seconds in *parameters.m*.
Gaussian distributed speed	The pedestrians' speed is normally distributed, computed in *initialisation.m*.

The probabilities defined in *parameters.m* are based on our subjective estimation.

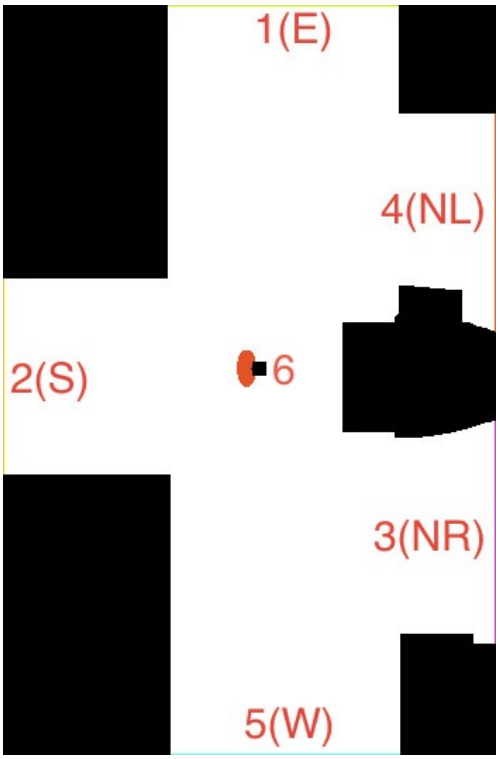

Figure 6: The simulation area with the five possible entries, or exits respectively.

After the initialisation of the agents, the programme enters the agent's loop and the individual forces are calculated: the destination force (*destination.m*), the walls force (*walls.m*) and the forces of the other agents (*other_agents.m*). The sum of the forces is computed and matrix *A* is updated (*int.m*). Within *int.m*, the function *exit2.m* is called: it deletes agents that have left the simulated area and saves the following parameters in matrix *C*: start time, entry and exit code number, paper-taker yes or no. The direction of those agents who have just taken a paper (i. e. who have spent the time it takes to pick up a newspaper inside the paper box area) is changed with *destination_change.m* to their final destination that was set in the beginning.

After the end of the agent loop, matrix F is updated: Matrix F is a matrix with same size as the map. It is first implemented as a null matrix. 1 is added to the

corresponding element every time an agent has been in a field. This matrix shows in the end, which places were occupied the most or the least respectively.

After the agent's loop is completed, the new situation is plotted (*plotter.m*) and the next time iteration starts.

4.3. Evaluation

For analysis (*finalanal.m*) of our pedestrian model the visual assessment is rather meaningless. Instead, we need the time data of every agent, which were saved in the C-matrix.

We ran 4 simulations for each of the 6 possibilities of paper box positions. One way to decide whether one box is suitable or not is a fairly general one and we used it for a first overview. It is the visualised matrix F, indicating the most critical places.

Secondly we calculated the median for different paths (namely every possible combination of the five exits with the differentiation whether an agent is a paper-taker or not). With that secondary data we had the tools to determine the most different situations to see if there is a noticeable effect in placing the box in different places.

5. Simulation Results and Discussion

A mere visual overview on the results is shown in Figure 7, the visualised F matrices of one run each. You can clearly see the area of the buildings and of the paper box (dark blue). Also, it can be seen that many agents have passed the corners and visited the area around the paper box (red). Additionally, light blue lines indicate walking lines of agents. The reddish regions around the paper boxes show how big the crowds were: In C the crowding around the box seems quite small, also in E, whereas especially D and F seem to have produced a bigger crowding.

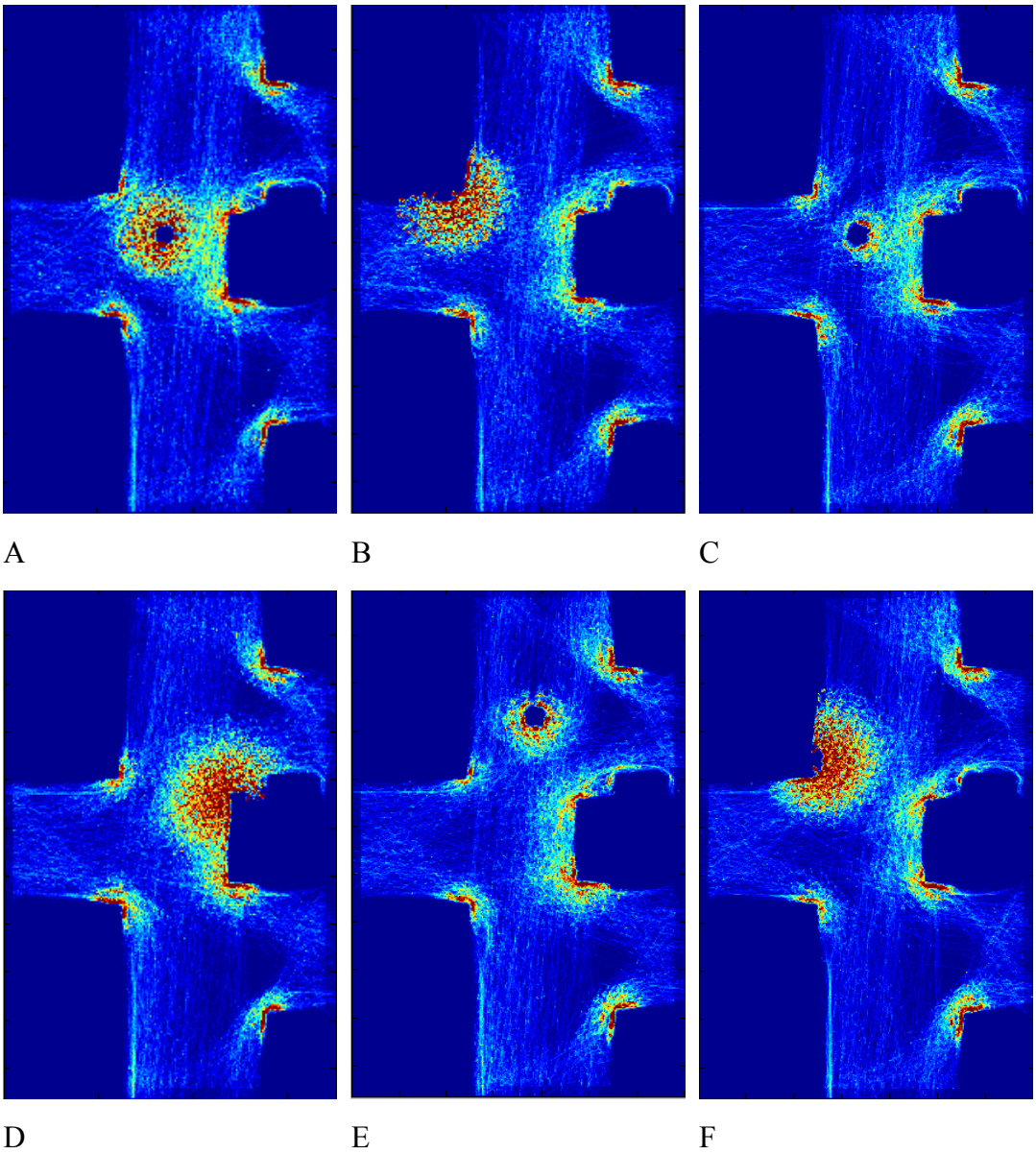

Figure 7: The visualised matrices F; colour scale: dark blue: No agent has passed the area, red: many agents have passed the area.

Table 1 shows the results in numbers: The median passing times in the different situations. If we look at the overall median passing times, the agents in C were the fastest, A was quite similar and the agents in D required the most time to pass the simulated area.

Table 1: The median passing times (in seconds) in the different scenarios, for all the agents, those who went to pick up a newspaper and those who did not.

	all	paper-takers only	without paper-takers
A	28.30	56.80	27.15
B	29.10	63.08	28.35
C	28.25	45.20	27.25
D	29.80	63.63	28.70
E	28.45	49.65	27.30
F	29.28	65.40	28.25

If we only take those agents into account, who picked up a newspaper, the shortest time required occurs in C again, and the agents in F were the slowest. The median passing times of the agents not interested in a newspaper are a little bit lower than the overall values, but the variations are quite similar. This is due to the fact that we are looking at the medians and that only a minority of the pedestrians picked up a newspaper.

Interesting is also the comparison between the situations A and C because the position of the paper box is the same, and only the orientation is different: The overall median is about the same in both situations, but the median passing time of the paper-takers is much higher in A than in C.

In the following, we will further discuss C versus D and C versus A.

5.1. C versus D

The most obvious difference between the situations C and D is that the agents picking up a paper require much more time in D than in C. As we can in Figure 7, the crowd around the paper box is much bigger in D than it is in C. So it makes sense that picking up a paper takes much longer in D. But why is the crowd bigger in D? We assume that this has something to do with the main pathways of the agents: In D, many of the non-paper-takers pass nearby the paper box in the edge and contribute to the crowding. This is especially the case in our model where agents just try to pass very close to the corners in order to find the shortest way possible. Whereas in C, where the paper box is in the middle of the free space, non-paper-takers can pass on the left or on the right.

From this picture we can conclude that a placement in the middle of a corridor is more favourable than a placement of the box at an edge. However, as will be

discussed in chapter 0, the pathways of the agents so close to the corners is not completely realistic.

5.2. A versus C

A comparison between situations A and C is also very interesting because the paper box is exactly at the same location. The only difference is its orientation: In A the box is pointing south (the direction where the train arrives) while in C it is pointing north. Looking at Figure 7 above we clearly see that situation A causes a huge bottleneck while the pedestrians in C are much better distributed. A has a much higher median for paper-takers but surprisingly, the median for non-paper-takers is about the same as in C.

We assume that in situation A most of the non-paper-takers are not significantly influenced by the bottleneck around the paper box since there is still enough space to walk around it. Of course some might get stuck but because we used the median outliers do not have a big influence. Paper-takers in A are likely to get stuck in the big bottleneck in front of the paper box. The bigger the crowd, the longer it takes to finally get to the paper box and also to get back out of the crowd. This “double negative” effect is also a reason why paper-takers need so much more time to get to the exit, especially if the bottleneck is big. Like in the comparison of C and D it is unsure whether this issue also applies to real pedestrian situations.

The orientation of C forces the paper-takers that come from the train to walk around the paper box and stay in the paper box area where non-paper-takers from the train do not pass. This causes a much better distribution of the agents, which improves the general pedestrian flow. Even if agents have a longer way to walk, the fact that the general flow is much better distributed leads to a faster solution for the median paper-taker. However we assume that this issue mainly applied because the number of agents and percentage of paper-takers coming from the train is higher compared to the other entries.

All the data can be found in the repository in code/analysis.

6. Summary and Outlook

After overcoming some problems with our implementation we finally got our code to run. Comparing the received results led to the surprising conclusion that situation C gives the fastest pedestrian flow. This is surprising as the paper box area points in the opposite direction of people coming from the train.

For a better evaluation of the results one should try to simplify the model to see which of these effects are real outcomes of the simulation or just some side effects of our implementation which occurs in every situation and has no real relevance for the actual result.

6.1. Difficulties

Generally it is very difficult to model human behaviour. Pedestrians calculate the walking direction of other people in their sight field in order to plan their own way. They avoid crowds and sometimes change their goal spontaneously. Also, when there is a huge crowd around a paper box, they might change their mind and just forego the newspaper. We did not include these behavioural factors in our model because we would have had to analyse people's behaviour very precisely, which would have gone beyond the scope of this project. A solution would be to use real time fast marching so agents can identify crowds as obstacles and walk around them. But since in our project we need to run the Fast Marching Algorithm 6 times for every exit plus the paper box, the simulation would be extremely slow.

Another difficulty we had to deal with was the repulsion force of the walls. When it was set too low, people wandered into the walls because the repulsive forces of the other agents was higher than the one of the walls. But when the repulsive force of the walls was too high, agents had problems to get into the paper box area, as the paper box was defined as a wall, because it is also something into which people cannot walk. However, after many trials we finally found quite good values for the wall repulsive forces.

Another problem with walls was that agents can get stuck in edges: This seems to be a problem of our implementation, because the repulsive forces in edges are not high enough to keep away the agents, especially when they are pushed by repulsive forces of other agents.

Finding good values was generally quite difficult, because little changes could influence the simulation a lot.

A main difficulty was also to interpret the results. We do not know whether our results are really relevant in real life situations or whether they are just results of

our implementation and the parameters we chose but have nothing to do with real human behaviour.

In order for our results to be more significant we would have to run the simulation different situations with changing parameters.

6.2. Suggestions for improvement

Looking back there are several things we would do different in our programme:

In order to overcome the problem of the agents who get stuck in edges, edges could be drawn as round shapes.

Walls could be programmed as outside the area so that agents just cannot walk into them.

The probabilities of pedestrians to pick up a newspaper, number of people coming from a train and entering and exiting through the different entries of Lucerne train station could be evaluated more scientifically rather than just being estimated freely.

A version of real time fast marching in order for pedestrians to avoid walking into crowds and get stuck could be implemented. The results with and without real time Fast Marching could then be compared.

7. References

Helbing, D. & Molnar, P. (1995). Social force model for pedestrian dynamics. *Physical Review E, 51*(5), 4282-4286.

Emch+Berger AG Bern. Simulation Fussgängerströme SBB Bahnhof Luzern. Retrieved 21. November 2012, from Emch+Berger AG Bern Web Site: http://www.bern.emchberger.ch/referenzen/1_13_move/simulation_fussgaengerstroeme_sbb_bahnhof_luzern

Heer, P. & Bühler, L. (2011). Pedestrian dynamics – airplane evacuation simulation. *Lecture with Computer Exercises: Modelling and Simulating Social Systems with MATLAB.*

Malladi, R. & Sethian, J. A. (1996). Level set and fast marching methods in image processing and computer vision. *International Conference on Image Processing, Proceedings, 1*, 489-492.

Peyre , G. (2008) Toolbox fast marching - A toolbox for fast marching and level sets computations. Retrieved 6. December 2012, from Matlab Central Web Site: http://www.mathworks.com/matlabcentral/fileexchange/6110-toolbox-fast-marching/content/toolbox_fast_marching/html/content.html

Vifian, M., Roggo, M. & Aebli, M. (2011). Modelling crowd behaviour in the Polymensa using the social force model. *Lecture with Computer Exercises: Modelling and Simulating Social Systems with MATLAB.*

8. Raw Data

Table 2: Raw Data table of our simulations.

		All	Takers	Not-Takers
	A			
#		1608	135	1473
median	1	576.5	1259	555
#		1599	131	1468
median	2	562	1054	543
#		1579	123	1456
median	3	558	1023	533
#		1588	119	1469
median	4	568	1289	545
#	Tot	6374	508	5866
median	Tot	566	1136	543
	B			
		1551	79	1472
	1	578	1038	566
		1531	74	1457
	2	576	1549	566
		1590	98	1492
	3	591	1314.5	571.5
		1553	99	1454
	4	583	1232	565
	Tot	6225	350	5875
	Tot	582	1261.5	567
	C			
		1620	118	1502
	1	557	832	540
		1609	142	1467
	2	597	961	565
		1599	124	1475
	3	556	899.5	536
		1614	123	1491
	4	559	939	541
	Tot	6442	507	5935
	Tot	565	904	545

D			
	1597	113	1484
1	590	1258	567
	1416	61	1355
2	586	1103	565
	1552	99	1453
3	618.5	1460	587
	1570	121	1449
4	598.5	1323	575
Tot	6135	394	5741
Tot	596	1272.5	574
E			
	1618	136	1482
1	580	985.5	559
	1604	121	1483
2	558.5	990	536
	1602	113	1489
3	560	935	543
	1585	139	1446
4	572	1080	549
Tot	6409	509	5900
Tot	569	993	546
F			
	1574	108	1466
1	581	1418	561.5
	1602	99	1503
2	589	1262	570
	1604	111	1493
3	594	1252	577
	1592	119	1473
4	574	1364	553
Tot	6372	437	59345
Tot	585.5	1308	565

9. MATLAB programme code

9.1. runfast.m

```
function [FOX,FOY,WX,WY,DistanceMap,ZB] = runfast()
%reads the painted picture into the programme and generates the
%gradientfield for each exit and the forcefield for the walls
%output: Gradientfield, directed to the exits (FOX,FOY)
%        Forcefield to simulate the wall rejection (WX,WY)
%        Value of the box_area (where the destination_change
                        happens)

%converts the picture into useable data
f=getmap();

%reads the value of the box_area
[ZB(:,1),ZB(:,2)]=find(flipud(f)==8);

%creates a matrix with only "wall=0" and "empty=1" entries
wall=find(f==0);
W=ones(size(f));
W(wall)=0;

%produces the forcefield for the walls
[WX,WY,DistanceMap]=grad_walls(W);

    for i=2:7 %iterates over all 5 possible exits
    [FX,FY]=computeGradientField(f,i);
       if(i==2);  %creates an empty matrix for the first run-
                        through
                [a,b]=size(FX);
                FOX=ones(a,b,6);
                FOY=ones(a,b,6);
        end
    %corrects the Y-axis inversion
    FX=flipud(FX);
    FY=-flipud(FY);

    %loop which reads the FX/FY values in a bigger R^3-matrix
    [a,b]=size(FX);
            for j=1:a
                      for m=1:b
                        FOX(j,m,i-1)=FX(j,m);
                        FOY(j,m,i-1)=FY(j,m);
                      end
            end
    end
end
```

9.2. getmap.m

```
function [F] = getmap()
%decode color coded map

[X,map]=imread('Bilder/PlanBhf_BoxB2.bmp');
I=im2double(X,'indexed');

[n,m]=size(I);
wall=find(I==1);
exit_west=find(I==7);
exit_south=find(I==8);
exit_northr=find(I==6);
exit_northl=find(I==5);
exit_east=find(I==3);
space=find(I==9);
box=find(I==2);
box_area=find(I==4);

F=zeros(n,m);
F(wall)=0;
F(space)=1;
F(exit_west)=2;
F(exit_south)=3;
F(exit_northr)=4;
F(exit_northl)=5;
F(exit_east)=6;
F(box)=7;
F(box_area)=8;

F=flipud(F);
imshow(F);
end
```

9.3. grad_walls.m

```
function [FX,FY,DistanceMap]=grad_walls(W)
parameters;
path('FastMarching_version3b',path);
%converts a matrix with (1,0) entries into a forcefield
%input:     the wall/space matrix W
%outputs:   forcefield (FX,FY) to apply the rejection
%           distancemap

 SpeedImage=W;
 [X,Y]=find(W==0);
 SourcePoint=[X';Y'];

 %Fast Marching Algorithm
 DistanceMap= msfm(SpeedImage, SourcePoint);
 DistanceMap=flipud(DistanceMap);

 %creates gradientfield
 [FX,FY]=gradient(DistanceMap);

 %normalises the Gradientfield
 s=size(FX);
   for i=1:s(1)
       for j=1:s(2)
```

```
            FX(i,j)=FX(i,j)/sqrt(FX(i,j)^2+FY(i,j)^2);
            FY(i,j)=FY(i,j)/sqrt(FX(i,j)^2+FY(i,j)^2);
        end
    end

%    for i=1:s(1)
%       for j=1:s(2)
%           FX(i,j)=FX(i,j)*U*exp(-DistanceMap(i,j)/R);
%           FY(i,j)=FY(i,j)*U*exp(-DistanceMap(i,j)/R);
%        end
%     end

end
```

9.4. computeGradientField.m

```
function [FX,FY] = computeGradientField( F,i )
 path('FastMarching_version3b',path);
%converts the in  getmap created situationmap into a Vectorfield
%using Fast Marching Algorithm and gradient* funktion.
%inputs: number of iteration, situationsmap
%outputs: values of the gradientfield in x and y direction

% legend for the matrix entries:
% wall=-1
% space=1
% exit_west=2
% exit_south=3
% exit_northr=4
% exit_norhtl=5
% exit_east=6
% box=7


%the i-th value is defined as exit
    Exit=find(F==i);
    F(Exit)=Inf;
    [rowE,colE,v]=find(F==Inf);

    Exits(1,:)=rowE';
    Exits(2,:)=colE';

%create a new matrix with only 1(space) and 0(walls)
  [sx,sy]=size(F);
  NewF=ones(sx,sy);
  wall=find(F==0);
  NewF(wall)=0;
  SpeedImage = NewF;

%conduct the Fast Marching Algorithm
%output: D=Distancemap which is used later on
  tic; [D, Y] = msfm(SpeedImage, Exits, false, false); toc;

%sets the walls equal zero again.
  D(D>600)=0;

%changes the distancemap to a gradientfield
  [FX,FY]=gradient_special2(D);
end
```

9.5. gradient_special2.m

```
function [FFX FFY] = gradient_special2(M)
%this function is a copy from the "Airplane_Evacuation-paper"
%modified for our own purpose

%special implementation of a gradientfield-computation
%   input: Distancemap M
%   outputs: normalised gradientfield (FFX,FFY)

[a b]=size(M);

FX=zeros(a,b);
FY=zeros(a,b);
caseX=0;
caseY=0;

%cases: W=Wall  ∞=Point to be handled
%  1.  W ∞ W
%  2.  W ∞ ∞


for m = 1:a    %y-direction
    for n = 1:b   %x-direction

        %x-direction
        if (M(m,n)~=0) %point is no wall element
            if(n>1 && n<b) %no element at the boarder of the
                        matrix
                if(M(m,n-1)==0 && M(m,n+1)==0)
                    caseX=1;
                elseif (M(m,n-1)==0)
                    caseX=2;
                elseif (M(m,n+1)==0)
                    caseX=3;
                else
                    caseX=4;
                end
            elseif (n<b) %element at the lower boarder of the
                        matrix
                if(M(m,n+1)==0)
                    caseX=1;
                else
                    caseX=2;
                end
            else %element at the upper boarder of the matrix
                if(M(m,n-1)==0)
                    caseX=1;
                else
                    caseX=3;
                end
            end


        else %point is a wall element
            if(n>1 && n<b) %no element at the boarder of the
                        matrix
                if(M(m,n-1)==0 && M(m,n+1)==0)
                    caseX=5;
                elseif (M(m,n-1)==0)
```

```
            caseX=6;
        elseif (M(m,n+1)==0)
            caseX=7;
        else
            caseX=8;
        end
    elseif (n<b) %element at the lower boarder of the
                matrix
        if(M(m,n+1)==0)
            caseX=5;
        else
            caseX=6;
        end
    else %element at the upper boarder of the matrix
        if(M(m,n-1)==0)
            caseX=5;
        else
            caseX=7;
        end
    end

end

switch caseX
    case 1
        FX(m,n)=0;
    case 2
        FX(m,n)=(M(m,n)-M(m,n+1));
    case 3
        FX(m,n)=(M(m,n-1)-M(m,n));
    case 4
        FX(m,n)=(M(m,n-1)-M(m,n+1))/2;
    case 5
        FX(m,n)=0;
    case 6
        FX(m,n)=0;
    case 7
        FX(m,n)=0;
    case 8
        FX(m,n)=0;
end

%FX(m,n)=caseX;


%y-direction
if (M(m,n)~=0) %point is no wall element
    if(m>1 && m<a) %no element at the boarder of the
                matrix
        if(M(m-1,n)==0 && M(m+1,n)==0)
            caseY=1;
        elseif (M(m-1,n)==0)
            caseY=2;
        elseif (M(m+1,n)==0)
            caseY=3;
        else
            caseY=4;
        end
    elseif (m<a) %element at the lower boarder of the
                matrix
```

```
        if(M(m+1,n)==0)
            caseY=1;
        else
            caseY=2;
        end
    else %element at the upper boarder of the matrix
        if(M(m-1,n)==0)
            caseY=1;
        else
            caseY=3;
        end
    end


else %point is a wall element
    if(m>1 && m<a) %no element at the boarder of the
                matrix
        if(M(m-1,n)==0 && M(m+1,n)==0)
            caseY=5;
        elseif (M(m-1,n)==0)
            caseY=6;
        elseif (M(m+1,n)==0)
            caseY=7;
        else
            caseY=8;
        end
    elseif (m<a) %element at the lower boarder of the
                matrix
        if(M(m+1,n)==0)
            caseY=5;
        else
            caseY=6;
        end
    else %element at the upper boarder of the matrix
        if(M(m-1,n)==0)
            caseY=5;
        else
            caseY=7;
        end
    end

end

switch caseY
    case 1 %  W ∞ W
        FY(m,n)=0;
    case 2 %  W ∞ ∞
        FY(m,n)=(M(m,n)-M(m+1,n));
    case 3 %  ∞ ∞ W
        FY(m,n)=(M(m-1,n)-M(m,n));
    case 4 %  ∞ ∞ ∞
        FY(m,n)=(M(m-1,n)-M(m+1,n))/2;
    case 5 %  I I I
        FY(m,n)=0;
    case 6 %  I I ∞
        FY(m,n)=0;
    case 7 %  ∞ I I
        FY(m,n)=0;
    case 8 %  ∞ I ∞
```

```
                FY(m,n)=0;
        end
        % current point: ∞    Wall: W   Infinity:I

        %FX(m,n)=caseX;

    end
end


%normalization
[a,b]=size(FX);

for m = 1:a
    for n = 1:b
        if (FX(m,n)~=0 && FY(m,n)~=0)
            FFX(m,n)=(FX(m,n)/(sqrt(FX(m,n)^2+FY(m,n)^2)));
            FFY(m,n)=(FY(m,n)/(sqrt(FX(m,n)^2+FY(m,n)^2)));
        elseif(FX(m,n)~=0)
            FFX(m,n)=(FX(m,n)/abs(FX(m,n)));
            FFY(m,n)=0;
        elseif(FY(m,n)~=0)
            FFX(m,n)=0;
            FFY(m,n)=(FY(m,n)/abs(FY(m,n)));
        end
    end
end


end
```

9.6. simu.m

```
function [F,C]=simu
clear all
close all
clc

load('a/FOX_FOY_WX_WY_DistanceMap_ZB.mat')  %load vector files
                        calculated by runfast.m
parameters; %load parameters.m

for i=1:size(WX,1)   %calculation of repulsion force of the walls
    for j=1:size(WX,2)
        WX(i,j)=WX(i,j)*U*exp(-DistanceMap(i,j)/R);
        WY(i,j)=WY(i,j)*U*exp(-DistanceMap(i,j)/R);
    end
end

r=0;
r_train=0;
A=zeros(0,11);
C=zeros(1,4);
c=0;

graycl=getimage;
s=size(WX);
F = zeros(size(WX));
```

```
vidObj=VideoWriter('simulation.avi');
vidObj.FrameRate=1/t;
open(vidObj);

for n=1:iter  %time iteration
    [A,r,r_train,c]=initialisation(A,r,r_train,n,c);
    i1=1;

    while i1 <= size(A,1) %agents iteration
        [FX1,FY1]=destination(A,i1,FOX,FOY);

        [FX2,FY2]=walls(A,i1,WX,WY);

        [FX3,FY3]=other_agents(A,i1);

         FX=FX1+FX2+FX3;
         FY=FY1+FY2+FY3;

         [A,C]=int(A,i1,FX,FY,s,n,C);

          A=destination_change(A,i1,ZB);

          i1=i1+1;
     end


      for z=1:size(A,1)    %update matrix F
            F(round(A(z,2)),round(A(z,1))) =
                        F(round(A(z,2)),round(A(z,1))) + 1;

    end

    plotter(A,graycl,n,c);
    currentframe=getframe;
    writeVideo(vidObj,currentframe);
end
close(vidObj);
end
```

9.7. parameters.m

```
%parameters

t=0.05;                 %delta t (seconds per second iteration)
iter=370/0.05;          %number of time iterations, simulated time
                        = iter*t
pxm=15;                 %pixel per meter

pers_per_s=3.25;        %new agents per second (except from agents
                        getting off the train)
train_delay=30;         %time until train arrives
pers_per_s_train=2.5;   %number of agents per second getting off
                        the train
nof_train=500;          %number of agents from train (total)


v0=1.3*pxm;             %average velocity of pedestrians
s_v0=0.3*pxm;           %standard deviation of pedestrians
```

```
                        velocity
arc=3/360*2*pi;         %angel of random rotation of velocity
                        vector
tend=6/360*2*pi;        %angel of rotation of velocity vector

Tau=0.5;                %parameter for force in direction of the
                        destination

A1=3*pxm;               %parameter for force of other agents
A2=3*pxm;
B1=0.1*pxm;
B2=0.2*pxm;
r_agents=2*0.3*pxm;
lambda=0.75;
sight_radius=2*pxm;

U=20*pxm;               %parameter for force of walls
R=0.3*pxm;              %0.2

pb_time=2.5;            %time to take a paper out of the box

p1_paper=0.04;          %probability of being a paper-taker from
                        agents of starting point 1
p2_paper=0.18;          %probability of being a paper-taker from
                        agents of starting point 2
p3_paper=0.04;          %...
p4_paper=0.04;
p5_paper=0.04;

p_1=0.25;               %probability to start at start point 1
p_3=0.25;               %probability to start at start point 3
p_4=0.25;               %...
%p_5=0.25;

p_12=0.1;               %probability of a pedestrian from start
                        point 1 to go to destination 2
p_13=0;                 %probability of a pedestrian from start
                        point 1 to go to destination 3
p_14=0.4;               %...
%p_15=0.5;

p_21=0.25;              %...
p_23=0.25;
p_24=0.25;
%p_25=0.25;

p_31=0.45;
p_32=0.1;
p_34=0;
%p_35=0.45;

p_41=0.45;
p_42=0.1;
p_43=0;
%p_45=0.45;

p_51=0.5;
p_52=0.1;
p_53=0.4;
%p_54=0;
```

9.8. getimage.m

```
function graycl=getimage()
%get the image for plotting

[X,map]=imread('Bilder/PlanBhf_boxA.bmp','bmp');
gray=ind2gray(X,map);
[a,b]=size(X);
for m=1:a
    for n=1:b
        if(gray(m,n)~=0)
        graycl(m,n)=255;
        end
    end
end
```

9.9. initialisation.m

```
function [A,r,r_train,c]=initialisation(A,r,r_train,nn,c)
parameters;

%initialises start position of new agents
%         A(i,1)     %start position x
%         A(i,2)     %start position y
%         A(i,3)     %speed x
%         A(i,4)     %speed y
%         A(i,5)     %current goal (1-6) =>inital goal(whether
                       final line or paper box)
%         A(i,6)     %time inside paper box area
%         A(i,7)     %Gaussian distributed velocity v0
%         A(i,8)     %final goal (1-6) (cannot be changed)
%         A(i,9)     %start time
%         A(i,10)    %code number of start position (1-6)
%         A(i,11)    %paper-taker yes (1) or no (0)

r=r+rand*2*pers_per_s*t;
    for i=1+size(A,1):size(A,1)+r-rem(r,1)
        start=rand;
        goal=rand;
        paper=rand;
        A(i,:)=0;

        if start<=p_1
        % start point 1
        A(i,1)=(128+271)/2+(rand-0.5)*(128-271);
        A(i,2)=10;
        A(i,10)=1;
        A(i,3)=0;
        A(i,4)=(v0+randn*s_v0);
            % goals different from 1
            if goal<p_12
                A(i,8)=2;
            elseif goal<p_12+p_13
                A(i,8)=3;
            elseif goal<p_12+p_13+p_14
                A(i,8)=4;
            else
                A(i,8)=5;
            end
```

```
    % paper-taker yes or no
    if paper<p1_paper
        A(i,5)=6;
        A(i,11)=1;
    else
        A(i,5)=A(i,8);
        A(i,11)=0;
    end


elseif start<=p_1+p_3
% start point 3
A(i,1)=340;
A(i,2)=(306+443)/2+(rand-0.5)*(306-443);
A(i,10)=3;
A(i,3)=-(v0+randn*s_v0);
A(i,4)=0;
        % goals different from 3
    if goal<p_31
        A(i,8)=1;
    elseif goal<p_31+p_32
        A(i,8)=2;
    elseif goal<p_31+p_32+p_34
        A(i,8)=4;
    else
        A(i,8)=5;
    end

    % paper-taker yes or no
    if paper<p3_paper
        A(i,5)=6;
        A(i,11)=1;
    else
        A(i,5)=A(i,8);
        A(i,11)=0;
    end


elseif start<=p_1+p_3+p_4
% start point 4
A(i,1)=340;
A(i,2)=(88+222)/2+(rand-0.5)*(88-222);
A(i,10)=4;
A(i,3)=-(v0+randn*s_v0);
A(i,4)=0;
    % goals different from 4
    if goal<p_41
        A(i,8)=1;
    elseif goal<p_41+p_42
        A(i,8)=2;
    elseif goal<p_41+p_42+p_43
        A(i,8)=3;
    else
        A(i,8)=5;
    end
    % paper-taker yes or no
    if paper<p4_paper
        A(i,5)=6;
        A(i,11)=1;
```

```
            else
                A(i,5)=A(i,8);
                A(i,11)=0;
            end


        else
        % start point 5
        A(i,1)=(130+272)/2+(rand-0.5)*(130-272);
        A(i,2)=522;
        A(i,10)=5;
        A(i,3)=0;
        A(i,4)=-(v0+randn*s_v0);
            % goals different from 5
            if goal<p_51
                A(i,8)=1;
            elseif goal<p_51+p_52
                A(i,8)=2;
            elseif goal<p_51+p_52+p_53
                A(i,8)=3;
            else
                A(i,8)=4;
            end
            % paper-taker yes or no
            if paper<p5_paper
                A(i,5)=6;
                A(i,11)=1;
            else
                A(i,5)=A(i,8);
                A(i,11)=0;
            end
        end


        % generally
        A(i,6)=0;
        A(i,7)=abs(A(i,3)+A(i,4));   % Gaussian distributed
                        velocity v0
        A(i,9)=nn;                  %start time
        r=rem(r,1);

    end


% agents from train

        if c<nof_train && nn*t>train_delay;  %waits until train
                        arrives (train delay) and initialises
                        maximal not_train agents
         r_train=r_train+rand*2*pers_per_s_train*t;
            for i=1+size(A,1):size(A,1)+r_train-rem(r_train,1)
            paper=rand;
            goal=rand;
            A(i,:)=0;
            c=c+1;
            % start point 2
            A(i,1)=10;
            A(i,2)=(205+325)/2+(rand-0.5)*(205-325);
            A(i,10)=2;
            A(i,3)=v0+randn*s_v0;
```

```
            A(i,4)=0;

            % goals different from 2
            if goal<p_21
                A(i,8)=1;
            elseif goal<p_21+p_23
                A(i,8)=3;
            elseif goal<p_21+p_23+p_24
                A(i,8)=4;
            else
                A(i,8)=5;
            end

            % paper-taker yes or no
            if paper<p2_paper
                A(i,5)=6;
                A(i,11)=1;
            else
                A(i,5)=A(i,8);
                A(i,11)=0;
            end

            % general
            A(i,6)=0;
            A(i,7)=abs(A(i,3)+A(i,4));   %Gaussian distributed
                        velocity v0
            A(i,9)=nn;                   %starting time

            r_train=rem(r_train,1);
            end
        end
end
```

9.10. destination.m

```
function [FX,FY]=destination(A,i,FOX,FOY)
parameters;

e(1)=FOX(round(A(i,2)),round(A(i,1)),A(i,5)); %vector to
                        destination A(i,5) at position
                        (A(i,1),A(i,2))
e(2)=FOY(round(A(i,2)),round(A(i,1)),A(i,5));

% random angle for rotating of vector
alpha=randn*arc+tend;
e=[cos(alpha) -sin(alpha); sin(alpha) cos(alpha)]*[e(1); e(2)];


FX=1/Tau*(A(i,7)*e(1)-A(i,3));  %force=1/tau*(v0*e-actual
                        velocity)
FY=1/Tau*(A(i,7)*e(2)-A(i,4));
end
```

9.11. walls.m

```
function [FX,FY]=walls(A,i,Gx,Gy)
parameters;
   FX=Gx(round(A(i,2)),round(A(i,1)));      %repulsion force at
                        current position
   FY=Gy(round(A(i,2)),round(A(i,1)));

   if isnan(FX)                  %if position is exactly between
                        two walls, force is equal to 0 instead of
                        NaN
       FX=0;
       FY=0;
   end

end
```

9.12. other_agents.m

```
function [FX,FY]=other_agents(A,i)
parameters;
FX=0;
FY=0;
v_n=[A(i,3);A(i,4)]/sqrt(A(i,4)^2+A(i,3)^2);

    for j=1:size(A,1)     %iteration for every agent
        if(i~=j)
        d=sqrt((A(i,1)-A(j,1))^2+(A(i,2)-A(j,2))^2);
            if d<sight_radius %calculates force only if distance
                        to the other agents is smaller than
                        sight_radius
                n=[A(i,1)-A(j,1);A(i,2)-A(j,2)]/d;
                phi=angle(complex(-n(1),-n(2)))-
                        angle(complex(v_n(1),v_n(2)));
                FX=FX+A1*exp((r_agents-d)/B1)*n(1)*(lambda+(1-
                        lambda)*(1+cos(phi))/2)+A2*exp((r_agents-
                        d)/B2)*n(1);
                FY=FY+A1*exp((r_agents-d)/B1)*n(2)*(lambda+(1-
                        lambda)*(1+cos(phi))/2)+A2*exp((r_agents-
                        d)/B2)*n(2);
            end
        end
    end
end
```

9.13. int.m

```
function [A,C]=int(A,i,FX,FY,s,nn,C)
parameters;

A(i,3)=A(i,3)+FX*t;        %dv=F*dt
A(i,4)=A(i,4)+FY*t;

if sqrt(A(i,3)^2+A(i,4)^2)>(1.2*A(i,7))   %limits maximum of
                        velocity to vmax=v0*1.2
    A(i,3)=A(i,3)/sqrt(A(i,3)^2+A(i,4)^2)*A(i,7)*1.2;
    A(i,4)=A(i,4)/sqrt(A(i,3)^2+A(i,4)^2)*A(i,7)*1.2;
end

    A(i,1)=A(i,1)+A(i,3)*t;    %dx=F*dt + x(t-dt)
    A(i,2)=A(i,2)+A(i,4)*t;

    [A,C]=exit2(A,i,s,nn,C);
end
```

9.14. exit2.m

```
function [A,C]=exit2(A,i,s,nn,C)
if A(i,1)<0.5 || A(i,1)>s(2) || A(i,2)<0.5 || A(i,2)>s(1)
            %if agent is outside the map – delete
    C=[nn-A(i,9) A(i,10) A(i,8) A(i,11);C];
            %write interesting parameters into matrix C
    A(i,:)=[];
end
```

9.15. destination_change.m

```
function A=destination_change(A,i,ZB)
parameters;

if
                        find(ismember(find(ZB(:,1)==round(A(i,2)))
                        ,find(ZB(:,2)==round(A(i,1)))))
            %if agent is inside the paper box area
    A(i,6)=A(i,6)+1;

    if     A(i,6)>=pb_time/t   %if agent has been inside the paper
                        box area for more than pb_time, change
                        destination to final destination
        A(i,5)=A(i,8);
    end
end
end
```

9.16. plotter.m

```
function plotter(A, graycl,nn,c)
parameters;
imshow(graycl)

hold on
plot(A(A(:,5)==1,1),A((A(:,5)==1),2),'o','color','b')
plot(A(A(:,5)==2,1),A((A(:,5)==2),2),'o','color','r')
plot(A(A(:,5)==3,1),A((A(:,5)==3),2),'o','color','g')
plot(A(A(:,5)==4,1),A((A(:,5)==4),2),'o','color','y')
plot(A(A(:,5)==5,1),A((A(:,5)==5),2),'o','color','k')
plot(A(A(:,5)==6,1),A((A(:,5)==6),2),'o','color','m')
text(10,15,['Time: ' num2str(floor(nn*t)) ' s'],'color', 'red')
text(10,40,['Agents: ' num2str(size(A,1))],'color','red')
text(10,65,['Pass.: ' num2str(c)],'color','red')
axis([0 350 0 531]);
hold off;
end
```