ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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

Project Report

Pedestrian dynamics in long, narrow hallways

Moser Manuel, Suter Yannick & Theiler Raphael

Zurich
December 2012

Agreement for free-download

We hereby agree to make our source code for 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.

Moser Manuel	Suter Yannick	Theiler Raffael

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of Originality

This sheet must be signed and enclosed with every piece of written work submitted at ETH.

I hereby declare that the written work I have submitted entitled

Pedestrian dynamics in long, narrow hallways

is original work which I alone have authored and which is written in my own words.*

Author(s)

Last name	First name
Moser	Manuel
Suter	Yannick
Theiler	Raffael

Supervising lecturer

Last name	First name
Donnay	Karsten
Balietti	Stefano

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (http://www.ethz.ch/students/exams/plagiarism_s_en.pdf). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

Place and date

Signature

*Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

Print form

Contents

1 Abstract

Authors: Manuel Moser, Yannick Suter, Raffael Theiler
Title: Pedestrian dynamics in long, narrow hallways

In this project, we want to have a closer look at pedestrians in narrow hallways, motivated by a situation at Zurich main station. To do this, we simulate pedestrians with Matlab who walk according to some rules. We managed our agents to pass each other by, to look ahead a few meters and to decide where to walk next.
At first, we wanted to have a closer look at the pedestrian flux when the number of persons per minute entering is increased, when there are clearly more people moving in the same direction against a few in the other direction. Next, we wanted to analyze the influence of aggressive fast people in a rush, slowly moving obstacles and the influence of drunkard (randomized walking) on the pedestrian flux.
But soon, our attention turned more to building/creating everything on our own and less about a fast simulation of different situations.
The main outcome of our simulations was that pedestrians tend to get stuck or create jams as soon as there are lots of people trying to pass the same hallway. The exact same hallway can work smoothly if there are not too many people, but jams can arise quickly. And once a jam starts, it often spreads out because people have to stop and walk slower.
Finally, we arrived at the following conclusions: To improve the pedestrian flux, broadening a hallway is a superb solution. Therefore, at our sitation scene at Zurich main station, it would be best if the hall users would try to leave more space for the pedestrians, and if the food shops Imagine and Nordsee wouldn't have chairs and tables outside.

2 Individual contributions

Manuel:	report, revising & commenting
Yannick:	logical functions, revising & debugging
Raffael:	graphical functions, visualization & agents

Listed above are only the main tasks everyone of us took care of, but we shared most of the work. The github commit report is not always mirroring the work behind those uploads because we often worked together on the code, debugged, commented and worked on the documentation while swapping laptops.

3 Introduction and Motivation

3.1 Motivation

As many commuters too, we all got annoyed by people rushing through the small corridor left in Zurich main station hall (the path between burger king and groups meeting point) during the Oktoberfest, market days, concerts and other occasions. So when thinking about a simulation project, this quickly crossed our minds. Special about this situation of a narrow but long corridor is that there's not much space to avoid other pedestrians to the side, especially if there are lots of other people whereas there is no need of path optimisation as the ultimate goal is to cross the hallway without crashing into someone else.

3.2 Fundamental Questions

The fundamental questions of our simulation project we started with are:

- We try to simulate the pedestrian flux of agents in a hallway in the following different situations: Rush hour (danger of jamming), much more agents in one direction than in the other, with an static obstacle (if possible), with aggressive/very fast or slow agents, random path agent (drunkard). Will the pedestrian flux run smoothly or will they block each other and be stuck?
- Will the implementation of a rudimentary kind of thinking/looking ahead help to avoid blockages? If possible, we may determine the limits for which the goal of passing is achieved with and without this implementation and compare them.
- Will there be group dynamics or similar behaviours of agents even if they're only programmed to walk to the other side, each on his own?

As soon as the programming phase started, we realized there was a major point of importance about this work we all were aware of, but had forgot to think about beforehand. We did not want to start with a simulation already created, but build something "new". So we started off creating our logic function that would allow the agents to avoid crashing into other agents. So, new fundamental questions arised:

- Starting with the idea of looking ahead, how can we turn this into a rudimentary kind of intelligence?
- How can we have a look at the main station situation but keep our agents as random as possible?

3.3 Expected Results

We think that there will be lots of changes in the direction of walking to the left and right while trying to avoid other agents. With rising amount of agents we expect more jams, this seems obvious. We think that in our simulation we'll have to deal with massive jams because the agents are not communicating with each other in any way. Our implementation of "looking ahead" as a form of intelligence will probably improve the people flux but only to a limited range.

4 Description of the Model

4.1 General considerations

As we wanted to describe a situation with people, we chose a model based on discrete agents. They should be able to walk freely along their path until they hit an obstacle, in which case they should be able to determine a way to avoid the obstacle. Obstacles could be walls, objects placed in the path or simply other agents. The main goal of any agent is to get to the other side of the path as quickly as possible, if possible without crashing into other things. As the global situation changes with each "step" an agents does and also with the appearance of new agents, a step-by-step iteration was chosen to propagate the situation in time in which the the optimal direction is calculated all the time. The other approach of calculating a certain path from start to finish was rejected as it probably cannot be done for the uncertainties mentioned above, namely the random appearance of new agents which could block the calculated path. Also, this is not a main point for our situation because there's not a specific destination an agent walks to, but a intended zone to go to. In a hallway without exits on the sides, the goal only is to traverse it.
Any agent's goal is to get to the other side as quickly as possible, although our model cannot accommodate the requirement of the quickest possible path. Because of the step-by-step iteration, any situation is repeatedly analyzed and (hopefully) the best way to proceed is chosen. As this is only a short time period and we only consider an agent's situation in a local environment, it cannot guarantee to give the best outcome overall. To make a long story short, we used a *local search algorithm.*

4.2 Walls and other static obstacles

If a narrow hallway should be a narrow hallway, it needs two walls. Although this statement is obvious, it can be implemented in various ways. We chose to use static agents with a small radius to act as a wall, as it allows the creation of many different hallways. They can also be used as static objects representing obstacles like chairs and tables which could stand in the path an agent has to follow to get to the other side. For our simulation, this was a very flexible way which also allows a quick adaption to other situations if necessary.

4.3 Agents

The basis of the simulation is the agent. An agent should represent a person in real life. We assumed that the hallway was not a place to linger about, therefore they should try to get to the other side as quickly as possible. In order to do so, they need to be aware of all the things around them that they might bump into. This was the origin of the thought that any agents only looks forward as the obstacles will be in the path before them. In our model, they also need some intelligence to get around an obstacle and avoid running into other agents. To do so, the agent should consider all things within a circle of defined radius in front of him while everything else doesn't bother him. Again, this tries to get a local solution to our problem in the hope that the overall solution is still a good one. As one usually doesn't walk backwards, our agents only walk forwards. This might not be true for all situations but for most of the situations a pedestrian walks into, and it simplifies the model considerably. We also think that, if a pedestrian does only walk more or less forwards and won't be involved in a jam or collision, his passing time will be quite good compared a perfect chosen path, if a such path even exists.

In comparison to previous models which used a force field to guide the agents, we thought that this approach is more realistic as the force field approach given that the force field already defines the optimal solution one is looking for. It should also prove to be more adaptable within a reasonable time frame to other problems.

4.4 Intelligence

4.4.1 General considerations regarding the necessary intelligence

For the model to work properly, a sensible solution to deal with the problem of finding the right path must be found. We do not claim to have found the best solution but believe our model to be a fairly good solution.
The presence of every other agent, be it agent or wall, inside a semi-circle in front of the currently considered agent clearly must have an influence in order not to bump into it. The fundamental idea is to get a function for every agent inside this agent's semi-circle which reproduces the effect it has on the path the considered agent would take without them being in his semi-circle. All of these functions will then be added and the maximum value would be taken as the best way to go forward. An additional function is added which should represent the tendency to go straight forward.

All functions were calculated as functions of x. This x is connected to an angle φ given by $arctan(x)$. A value of $\varphi = -\pi/2$ would correspond to walking sideways to the left, $\varphi = 0$ to walk straight forward and $\varphi = \pi/2$ to walking sideways to the right, always viewed in the direction the agent wants to go.
A function calculated in x will then be superimposed on φ which corresponds to a mapping of the function onto a polar grid. A possible disadvantage of this procedure is that φ does not range from $-\pi/2$ to $\pi/2$ and therefore doesn't reach the full semi-circle as x is bounded and doesn't reach infinity. We consider this to be negligible as

for reasonably high values of x, φ gets close to $\pm\pi/2$ and it is rarely in the interest of an agent to walk purely sideways.

For an agent coming from the other side, we used a function of the type

$$y(x) = \frac{1}{(|x - x_{\text{Agent}}|)^n} \tag{1}$$

and set the y values corresponding to x values on a collision course to zero. The exact implementation will be treated in chapter *Implementation* where equation (1) is modified to accommodate for various different parameters. If an agent has another agent in front of him which is walking in the same direction but slower, the function given in (1) is also applied to overtake the slower agent.
Should another agent walk in the same direction, but with higher speed, we thought that it would be logical to follow him. This was done using a Gaussian curve centered on the direction of the faster agent.
Agents moving in the same direction with the same speed have no influence on each other in our model. Two agents standing still on the other hand will also be handled using the function given in (1) to avoid standoffs.

4.4.2 Necessary parameters and variables

We reckon that the important parameters necessary to assess the influence of one agent onto the other agents are

- α_X, the angle between the centers of the two agents with respect to the y-axis
- Δv, the difference in velocity, set to be negative for this situation
- β_{Links} and β_{Rechts}, two angles describing which directions of the path to be chosen would lead to a collision.
- $r_S = r_1 + r_2$ being the sum of the two involved agent's radii
- d, the distance between the two involved agents

In figure 1 (given in the implementation chapter) a graphical depiction of α_X, β_{Links} and β_{Rechts} is given alongside the way to calculate them.
The sign of Δv was chosen to be positive in the case of two agents walking in the same direction and negative in the case of crossing agents.

4.4.3 Collision detection and the its connection to the iteration speed

Collision detection has to be carried out in order to keep the simulation physically possible. We used an approach in which the vector describing the path the agent walks on is subdivided into several small steps. The agent then goes on as far as he can. To avoid the agents sticking too close together and therefore not being able to move anymore, the number of subdivisions should be rather small.
Given that the length of the vector is determined by the iteration time intervall Δt, it is also important for Δt to be not too small. This has a phyiscal representation as in reality one goes on in discrete steps, not infinitesimally small steps that one would get in the case of $\Delta t \to 0$.

4.5 Discretization

As it is a numerical calculation, at some point there has to be a numerical discretization. Since we wanted the agents to be able to move around freely, we abandoned the idea of using a grid on which the agents could walk around in favour of a continuous space simulation. The discretization comes about in the form of the functions and angles to be calculated.

4.6 Spawning of new agents

A hallway usually has at its two endings a more open space than the hallway where the flow of people arriving from the hallway will disperse. In order for things not to get too messy, we decided that agents would simply cease to exist if they reach a line (called dstruction line) which would signify the beginning of this region of dispersion. Newly spawned agents would be created on a line (called spawn line) below the destruction line and pass a short distance without oncoming agents so that they could already orientate themselfes according to the flow of agents walking the other way. This could be important in cases where the agents would form columns and newly spawned agents could adapt to that by not interrupting the column.

5 Implementation

5.1 General considerations

We had to build our implementation around several different desires:

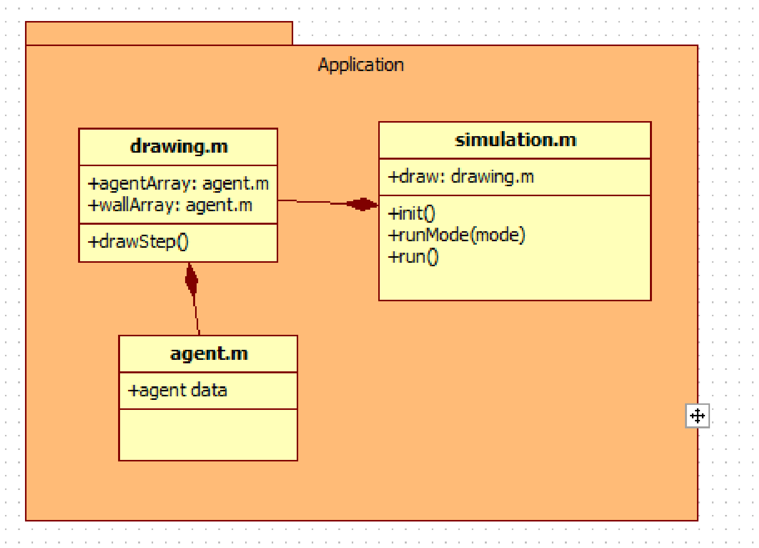
- Code has to be expandable with additional features.
- If there is a class model it has to be simple because the project is not that big.
- Constants should be easy to find.

According to those points the simulation code is a mixture between classes and cascaded functions. At the top is a file defining all the global constants. This approach is maybe not very correct in therms of a good programming style but it makes it very simple to adjust smaller details.

There are 3 classes: simulation.m, agent.m and drawing.m .The further description about these classes can be found in the table or direct in the header of the class files. The rest of the code is split up into different logical functions.

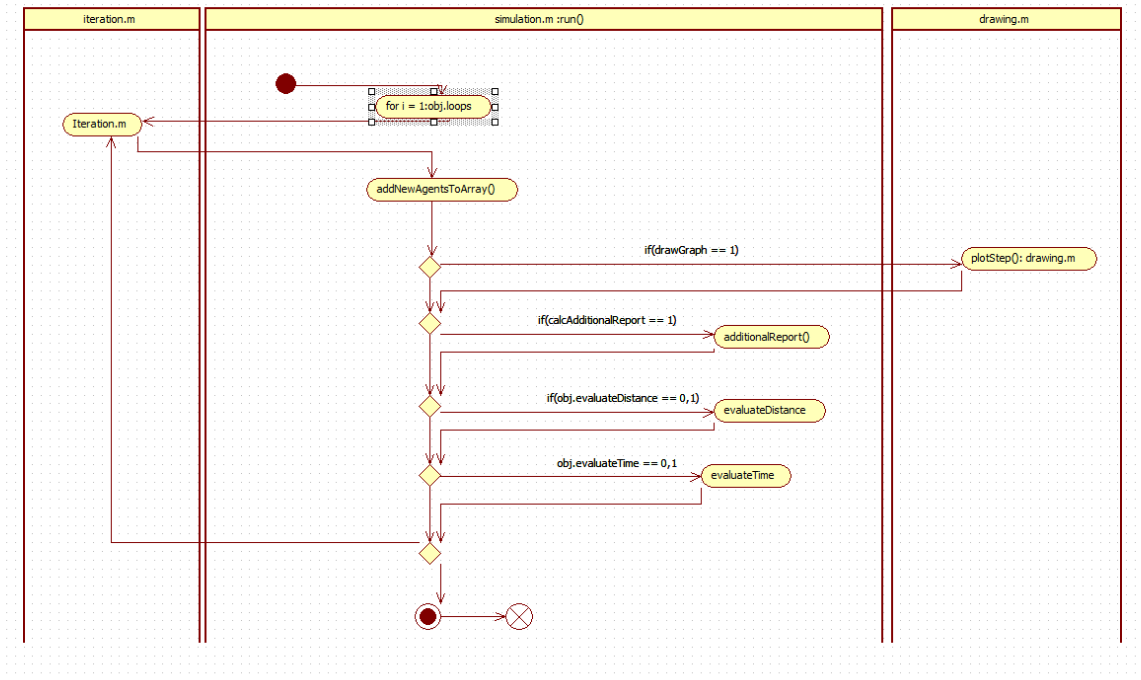
Class	Description	Parent
simulation.m	This class offers all the functions for simulations. It can be executed with different parameters depending on the users preferences.	Matlab "handle" class
agent.m	This class is a container for all the agent data such as its radius and the coordinates.	Matlab "handle" class
drawing.m	This class can draw a field containing agents.	Matlab "handle" class

The following image is a summary of the class relations containing the most important functions and properties:



5.2 simulation.m

The simulation class describes an object that wraps all the different possibilities of our simulation program. Its the start point for a new simulation, runs this simulation and collects all the data requested from the agents and the simulated environment. The main flow of a run cycle is described in the following activity diagram:



5.2.1 Properties

Important properties:

- draw: The drawing.m implementation.
- result: A matrix containing simulation results
- additionalResult: A matrix containing simulation results
- evaluateTime
- evaluateDistance

5.2.2 Methods

All methods are listed here:

- simulation(): Constructor, sets up the object.
- init(): Initializes the existing object: Fills up arrays with object, etc.
- runMode(): Pass console like parameters to this function to run a simulation.

- additionalReport(): Fills the additionalResult matrix.
- calcPossibleAgents(): Calculates the maximum possible number of agents for the set area.
- randPrefix(): Randomly generates 1 or -1 to set the spawnpoint (top or bottom).
- initialSPawn(): Fills up the agent array with new agent.m objects. They have priority 0.
- addNewAgentsToArray():Used to add new agents to the simulation while running.
- balanceProbability(): Balances the probability to spawn agents.
- **run()**: Main function to run a simulation after everything is set up and ready. See the activity diagram for a better explanation.

5.3 drawing.m

This class basically adapts simple Matlab drawing functions and converts them into useful functions for this project. As a result its very easy to draw the new situation after a simulation step.

5.3.1 Properties

Important properties:

- particleDensity: Resolution for wall agent.m objects in $\frac{agents}{meter}$.
- width: The field width.
- length: The field length.
- wallArray: All the agent.m object for the wall.
- agentArray: All the agent.m objects for the simulation.

5.3.2 Methods

All methods are listed here:

- drawing(): Constructor, sets up the object.
- createWall(): Creates wall agents according to the settings.

- **plotStep()**: Main function, plots all the agents on a field with the walls and the starting lines.
- drawWallSquares: Draws the walls on the side.
- circlePlot: Draws circles for the agents.
- drawLine: Draws a line with coordinates. Used for the direction indicators etc.

5.4 Agents

As the model is agent-based, we wanted to implement them as such. Therefore we created a class called agent.m. Every agent has the following properties:

- Radius of the agent called *radius*
- An x coordinate called *cordX*
- A y coordinate called *cordY*
- A maximal velocity called *maxSpeed*
- An actual velocity called *actSpeed*
- A priority called *priority*

A circle is the mathematically easiest shape to consider especially for collision detections which have to be done later all the time. In addition, we consider the circle to be a good approximation as one also needs some space to move as the legs cover some space in front and behind the body. A circle is also practical as it only needs one parameter entirely for the shape which is the radius. Using this approach, every agent can have a different radius.

The coordinates of the agent with respect to a cartesian grid centered at the lower left of the whole filed are vital for all calculations. They are also needed to defined the circle representing the agent in the plane. After each iteration, they will be adjusted to the new situation.

Every agent has a maximum velocity. The actual velocity of an agent gives its actual speed. If there is no obstacle, it will be the maximum velocity. In the initialization of an agent, the first actual velocity is set to be equal to the maximum velocity. As velocity is in principle a vectorial quantity, we used the sign of the maximum velocity to determine the way an agent walks. By our own convention a positive sign means to go upwards and a negative sign to go downwards.

Every agent has a priority and usually a random number between 0 and 1. It is used to determine the order in which the agents move through one iteration step. A high priority means that the agent will walk first. A high priority value can be attributed to an agent which will always try to push his way through.
The priority is also used to mark inactive agents. As all arrays are stored in an array of class agent, a priority of 0 denotes an empty position and will not be drawn are considered. Any new agent is placed at the first position in the array of agents with a priority of 0. If an agent reaches its goal, it can be deleted by setting the priority to zero. This is the only time the priority value is changed after the creation of an agent.

The class agent was implemented using *handle* in the class definition. This has the same effect as *call by reference* in C++ as there is after the creation only one instance of the agent, specially in the case where it is passed down to functions. A change of the value inside the function will be effective also outside the function which saves the step of giving the whole array of agents back after every function call which changes the coordinates or the actual velocity.

5.5 Drawing of the field

5.6 Logical functions

5.6.1 General considerations

The heart of the simulation is the function *logicFunction.m*, which determines the path any agent will choose to get to the other side of the hallway. To be more precise, it determines only the next step an agent will take and not the whole path. It relies heavily on the two functions *xValuesLogic.m* to deal with other agents and *xWallLogic.m* to deal with agents representing the wall or static obstacles. At first, the functioning of *xValuesLogic.m* will be explained and afterwards the functioning of *xWallLogic*.

5.6.2 How to get β_{Links} and β_{Rechts}?

For our model, it is crucial to determine where an agent shouldn't go. The function *getBeta.m* returns the angles which describe the interval leading to a collision. A graphical depiction of the situation is given in figure 1. The equations (2) to (4) were used to get β_{Links} and β_{Rechts}. They had to be converted into the angles given with

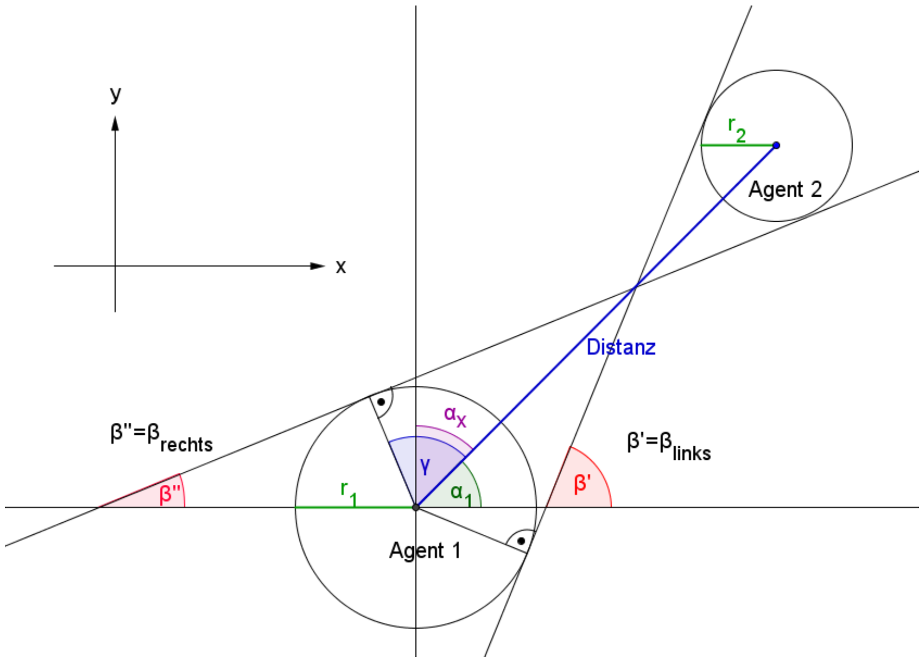


Figure 1: The graph shows the angles and variables used to get β_{Links} and β_{Rechts}. α_X is the angle between the two agents with respect to the y-axis. This depiction was engineered to work also for agents walking the other way.

respect to φ, $\beta_{\varphi,\text{ left}}$ and $\beta_{\varphi,\text{ right}}$ as shown in equations (5) to (6).

$$\gamma = arccos\left(\frac{r_S}{d}\right),\ \ \alpha = arctan\left(\frac{\Delta y}{\Delta x}\right) \tag{2}$$

$$\beta_{\text{Links}} = \gamma + \alpha - \frac{\pi}{2} \tag{3}$$

$$\beta_{\text{Rechts}} = +\alpha + \frac{\pi}{2} - \gamma \tag{4}$$

$$\beta_{\varphi,\text{ left}} = \frac{\pi}{2} - \beta_{\text{Rechts}} = \pi - (\gamma + \alpha) \tag{5}$$

$$\beta_{\varphi,\text{ right}} = \frac{\pi}{2} - \beta_{\text{Rechts}} = \gamma - \alpha) \tag{6}$$

This works between agents as well as between agents and the wall agents. Care was taken to engineer a calculation that allows for it to be used for agents walking in both directions.

5.6.3 xValuesLogic.m

xValuesLogic.m distinguishes three different cases.

- For two crossing agents or if the agent in front of the agent in question is slower, we used equation (7) to get x'_{out}. It was also used for two not moving agents, setting Δv equal to an arbitrary value given in *STANDOFF*.

$$x'_{\text{out}} = \frac{1}{(|x - \alpha_X|)^{\left(\frac{-\Delta v}{a}\right)}} = (|x - \alpha_X|)^{\left(\frac{\Delta v}{a}\right)}, \; \Delta v < 0 \tag{7}$$

  All values which correspond to a collision course in $x_y out'$ are set to zero. This also deals with the singularity of equation (7) as it is set to zero. This done using the β-angles shown before. Afterwards, x'_{out} is normalized and modificated further using equation (8).

$$x_{\text{out}} = x'_{\text{out}} \cdot \frac{b}{max(x'_{\text{out}})} \cdot \left(\frac{r_S}{d}\right)^c \tag{8}$$

  The variables a (called *SLOPEFACTOR*), b (*HEIGHT*) and c (*REPULSION-AGENT*) have to chosen in a way that the simulation runs smoothly. The term $\frac{b}{max(x'_{\text{out}})}$ normalized the function to a maximum value b while the term $\left(\frac{r_S}{d}\right)^c$ controls that the repulsive influence gets stronger the closer the two agents get. c is usually chosen to be larger than 1.

  If the x_{out} given in equation (8) would be returned, the agent in question would aim to miss the other agent exactly. We thought that this would be to close as in reality, one also leaves a bit of space if possible between each other. Therefore we introduced an offset given as *WALLANGLEOFFSET* which gives the angle additionally to the β angles for which an agent should aim to. To account for this, x_{out} is modified with an linear interpolation between the values at $\beta + WALLANGLEOFFSET$ and β (which was set to zero before).

- If the agent in front of the agent in question is faster, a gaussian curve was used with the mean α_X and standard deviation rS/d. It is then modified further with Δv and *HEIGHT* to make it a weak influence.

- For two agents moving with the same speed, the influence is set two zero by returning a vector of zeros.

5.6.4 xWallLogic.m

To avoid hitting the wall, we used a very simple approach. Every angle corresponding to a collision course is set to a negative value accoring to equation (9).

$$x_{\text{Out}} = x \cdot \frac{a}{d - rS} \tag{9}$$

As before for agents, an offset is introduced so the agent in question doesn't just try to avoid the wall-agent but also to leave some buffer space. The offset is also given in *WALLANGLEOFFSET*, a can be accessed with the constant variable *WALLFACTOR*. a has to be set negative as otherwise the wall would have an attractive force. To set a good value for this factor a is quite delicate because if it is too low agents will be stuck in the wall while if it too high they will never approach the wall even slightly.

5.6.5 Calculating functions in x and transforming them into a polar axis in φ

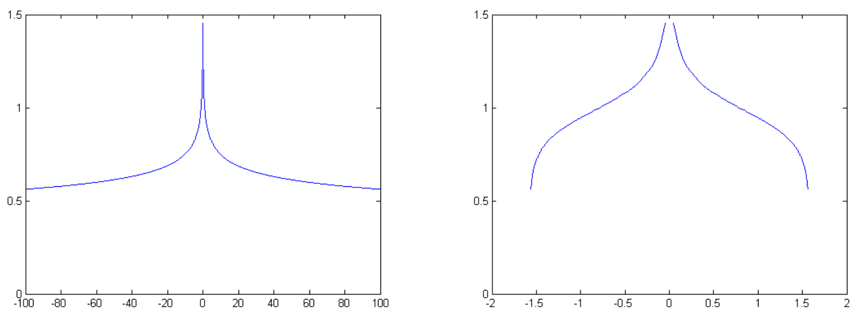


Figure 2: Graphical depiction of the function given in equation (7). In the graphic on the left, the horizontal axis is given in x while in the right graphic the horizontal axis is given in φ.

As the functions given above are given in x but the direction to go on is determined in polar values, it needs to be transformed into a φ axis. This is done using a vector for x ranging $\pm$ *XSCOPE* with a step of *XRES*. Applying the arcustangent on it yields the axis in angular values (φ). The transforming is done simply by using the φ axis instead of the x axis. This is shown in the two figures 2 and 3. As those functions are only chosen to show the principle, they were not normalized in height according to the equations mentioned above.
In figure 2, the function (7) is shown graphically in the x as well as φ axis. Figure 3 shows the x_{out} the function *xValuesLogic.m* returns.

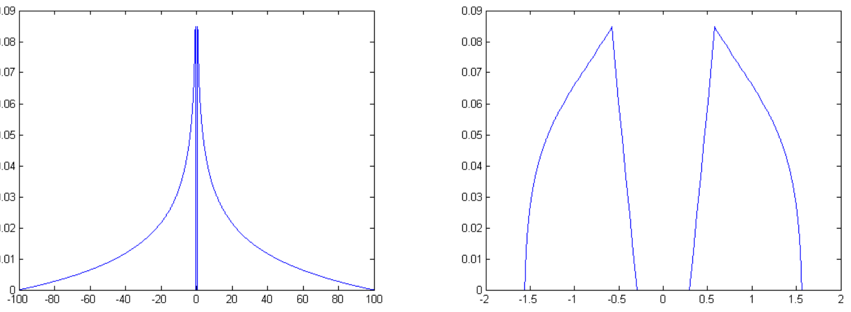


Figure 3: Graphical depiction of the return values of the function *xValuesLogic.m*. In the graphic on the left, the horizontal axis is given in x while in the right graphic the horizontal axis is given in φ. α_X was set to 0 corresponding to an other agent directly ahead, β was set to ± 0.3 with an offset of 0.25.

5.6.6 Graphical example

This subchapter shall give a visual explanation of how the logic functions work. Lets consider the situation given in **??**.

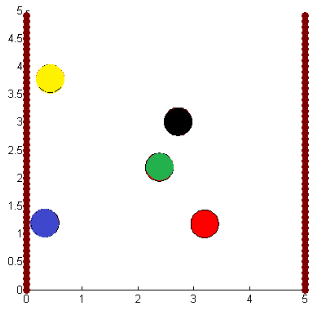


Figure 4: Exemplary case used to demonstrate the working principle of the logic functions.

The blue agent moving up in figure 4 only sees the influence of the wall. What the blue agent ”sees” is given in figure 5. The influence of the wall causes the overall function to decrease for all φ corresponding to a collision course. The offset causes the overall function to have its maximum α at a positive φ. This causes the agent to walk in the direction of α.

The green agent moving up sees only the influence of the black agent who is moving down. The effect of that it given in figure 6. The underlying gaussian function can

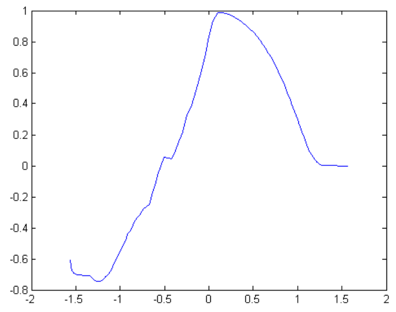


Figure 5: Output of all logic functions combined for the blue agent in figure 4. Visible is the effect of wall agents on the blue agent as negative values on the left side.

be seen as well as the addition of a modified version of figure 3, left. The agent will move slightly to the left to avoid hitting the black agent.

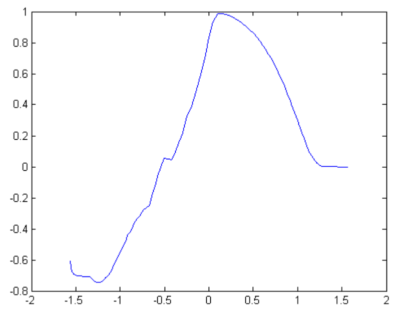


Figure 6: Output of all logic functions combined for the green agent in figure 4. Visible is the effect of the oncoming black agent as a superposition on the underlying gaussian curve.

The black agent moving down sees the oncoming green and red agent going up. The effect of them is given in figure 7. The superposition of two functions onto the underlying gaussion can be seen by the discontinities. The effect of the green agent is stronger as it is nearer to the black agent than the red agent which causes the black agent to go left (looking top-down) in order to avoid hitting the green agent. This shows the dependency of the strength of an agent of the distance.

5.7 Iteration

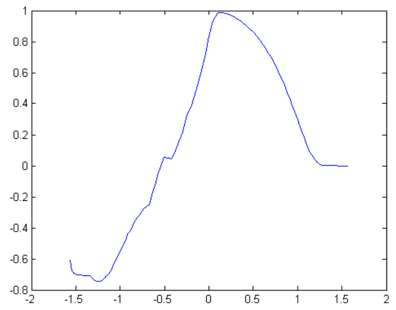


Figure 7: Output of all logic functions combined for the black agent in figure 4. Visible is the effect of the oncoming green and red agents as superpositions on the underlying gaussian curve. The black agent will move left (from his point of view) to avoid hitting the closer green agent.

5.8 Simplifications

- x All our agents walk on their own, there are no groups of friends, families etc who stand together as much as possible.
- x The agents are not able to walk backwards, they only can see and walk $90°$ to each side.
- ? All have the same mean speed, size, ??
- ? Any other simplifications?
- x Feld gekürzt, da Verhältnis 60:2.8 schwierig auf Bildschirm

6 Simulation Results and Discussion

6.1 Goals

First, let's have a look at what our goals were. We planned to have a look at the pedestrian flux, how it can be improved and jammings be avoided. We furthermore wanted to have a closer look to what happens during rush-hours and in a situation when much more people are moving in one direction than in the other.
On the agent-based side of our model, we wanted to analyze the influence of aggressive fast people in a rush, slowly moving obstacles (eg. mothers with baby buggies) and the influence of drunkard (more or less randomized walking) on the pedestrian flux.
If everything went well, we also wanted to implement a static obstacle and see what happens.

6.2 Achievements

As soon as we started programming we realized there was a major point of importance about this work we all were aware of, but had forgot to put it in the project proposal. We all did not want to start with an already known program or algorithms, but build something "new". So we started off creating our logic function that would allow the agents to avoid crashing into other agents and not working with repulsive forces as for example Helbing (Quelle angeben, ist das älteste Paper) did.
Quite proudly, we can now say we managed to do this. Our idea of the agents "thinking ahead" by consulting where other agents are and not just being pushed around by repulsive forces worked.
We now are able to play with lots of input variables, the most important being number of agents entering the corridor per time and the agents' characteristics as size, speed and lots more.
A nice thing we built but did not originally plan to is that we planned to and did research on the sitation as explained earlier in the long, narrow corridor in Zurich mainstation. But in our simulation, one can also change dimensions as length and shape of the walls easily.

6.3 Fundamental Questions

Our fundamental research questions were:

- How does the simulation behave in the following situations: rush hour, with obstacle, with very slow/fast agents, random path agent (drunkard)? Does it run smoothly or will ther be jams?

- How will our implementation of a rudimentary kind of ?thinking ahead? affect the simulation? Will it work good or bad? Can we compare it to other implementations?

- Are there any group dynamics evolving as lane or group formation?

6.4 Comparing measurements

Saturday, Nov 17th, we did some quick mesaurements right at Zurich main station to have some data we could compare. Two measurements were taken, only some minutes lay between these, that was when we measured the length and breadth of our corridor. The measurements were:

1 The "boring" measurement: During 2 minutes 14 pedestrians headed tracks 3-18, and 20 pedestrians directed towards tram station "Bahnhofsquai". No problems at all, very fluently.

2 The "crowded" measurement: During 2 minutes, 41 pedestrians headed tracks 3-18, and 33 pedestrians directed towards tram station "Bahnhofsquai". People got stuck, ran into each other, had to walk stop-and-go-like.

7 Summary and Outlook

8 References

9 Want-To-Do-List

! OUTPUT :

- Untersuche Situationen wenn: Anz. Agents variiert, Gangbreite variiert.
- Plot mit: x-Achse Zeit, y-Achse Personen pro Zeitabschnitt (evtl. mehrere deltaT) erzeugt/gelöscht, je oben& unten, für verschiedene Gangbreiten
- Weglängen? (Problem: abhängig von Anz. Agents im System, und diese ändert sich immer wieder
- Jeder Agent: gesamthaft gelaufene Strecke pro deltaT
- Wie viele Agents im System drin?
- Output automatisch speichern?

! TEST:

- Mit Dispersionfactor rumspielen

! DOKUMENTATION :

- Für alle Funktionen einen Kurzbeschrieb erstellen mit: Sinn und Zweck, Rechenmethode, evtl noch Kommentare dazu.
- Code auskommentieren!
- Kapitel "How to start our simulation" und was man wie wählen kann, ebenfalls run auskommentieren
- Kapitel Limitations of our model (dort simplifications reinnehmen, 180°-limitation verhindert "ausbrechen" (Bild), etc.

! INFO zu simulation results:
The following informations can be obtained through the simulation file:

- **sim.loops**: Number of loops calculated
- **sim.evaluateDistance**: ($1\times$(# of arrived agents))-matrix containing the distances that the arrived agents walked

- **sim.evaluateTime**: (1×(# of arrived agents))-matrix containing the time it took the arrived agents to arrive
- **sim.calcAdditionalReport**: 1, if the additional results were chosen, else 0
- **sim.spawned**: (2×loops)-Matrix containing the number of spawned agents at the top (column 1) and bottom (column 2)
- **sim.result**: (2×loops)-Matrix containing the number of deleted agents at the top (column 1) and bottom (column 2)
- **sim.additionalresult**: (2×loops)-Matrix containing the total distance walked by all agents at each iteration step (column 1) and the number of agents in the system (column 2)
- **sim.drawGraph**: 1, if a graphical output was drawn, else 0

A Appendix

MATLAB HS2012 - Research Plan

Version info: the submitted and approved version, 2012-10-24 17h

- **Group Name**: Mayara
- **Group participants names**: Moser Manuel (Mathematics BSc, 3rd Sem), Suter Yannick (Chemistry BSc, 5th Sem), Theiler Raffael (Informatics BSc, 3rd Sem)
- **Project Title**: Pedestrian dynamics in long, narrow hallways

General Introduction

Annoyed by people rushing through the small corridor left in Zurich main station hall (the path between burger king and groups meeting point) during the Oktoberfest, market days, concerts and other occasions, we decided to have a look at pedestrian dynamics in hallways which are mostly crowded and narrow (3-4 meters in breadth) compared to normal days when the hall is empty, and where people walk through in opposite directions all the time. We want to have a look at how the pedestrian flux can be improved and how the walk-through time behaves during rush-hours, but also in the case of much more persons moving in one direction than in the other. Also, we want to have a look at the influences of aggressive, fast people in a rush, slowly moving obstacles like mothers with baby buggies and some drunkards, and try to figure out how to avoid jammings. Maybe we'll also implement a static obstacle to observe what happens. The simulation of problems like this will also help understand the phenomena of group dynamics which usually control and resolve such problems in real life.

The Model

We want to do an agent-based simulation of people moving through a long corridor (dimensions will be proportional to those encountered in our object, the Zurich main station). The people will primary want to move forwards at different speeds but also be able to move diagonally or even sideways if needed. A nice thing will be to try implementing agents being able to see some fields/meters ahead whether their path (assumed straight as long as possible) is free or not, and if they're about to crash into someone, try to avoid them. Independent variables in our model are the amount of people per time arriving, the corridor and its obstacles and the characteristics of the agents like walking speed and aggressiveness. Dependent variables will be the amount of people leaving, which should in the end determine whether the people will be stuck or if they can get through. Should the amount of people leaving be smaller than those arriving (per time unit), one can expect a blockage. As a reference, we will use a simulation of a corridor without any obstacles and only few people. Then the collective success or failure of an other situation can be compared to this.

Fundamental Questions

- We try to simulate the pedestrian flux in the following different situations: Rush hour (danger of jamming), with an static obstacle, with aggressive/very fast or slow agents, random path agent (drunkard). Will the pedestrian flux run smoothly or will they block each other and be stuck?
- Will the implementation of a rudimentary kind of thinking/looking ahead help to avoid blockages? If possible, we may determine the limits for which the goal of passing is achieved with and without this implementation and compare them.
- Will there be group dynamics or similar behaviors of agents if they're only programmed to walk to the other side, each on his own?

Expected Results

We think that there will be lots of walking around left/right while trying to avoid other agents, and with rising amount of agents there will be more jams, this seems obvious. We think that in our simulation we'll have to deal with massive jams because the agents are not communicating with each other in any way. Implementation of "looking ahead" will probably improve the people flux but only to a limited range. Obstacles will also lead to more jams, whilst the drunkard simulation will for the amusement of our group.

References

Just some ideas where to get inspiration from:

- Project Suggestions - 16 - Pedestrian Dynamics - 5 papers - http://www.soms.ethz.ch/teaching/MatlabFall2012/projects/16-Pedestrian_Dynamics.zip - (01.10.2012)
- Mehdi Moussaid Publications - http://mehdimoussaid.com/publications.html (24.10.2012)

- Crowd-Flow-Optimization - FS2012 - https://github.com/nfloery/crowd-flow-optimization (01.10.2012)
- Train Jamming - FS2011 - https://github.com/msssm/Train_Jammin (01.10.2012)
- Airplane Evacuation / FS2011 https://github.com/msssm/Airplane_Evacuation_2011_FS (01.10.2012)

Research Methods

For our project, an agent-based model is the most satisfying because there we can really implement different speeds and directions. A disadvantage will be the complicated collision handling.

Other

For the measurements of the corridor, we'll go to the main station and measure it one afternoon when it's not fully crowded. We also could count the rate of incoming and leaving people during a rather relaxed afternoon and a crowded rush-hour.