ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Complex Social Systems:
Modeling Agents, Learning, and Games

Project Report

AI in Pedestrian Dynamics

Michael Eichenberger, Isabel Heidtman, Alexander Jürgens,
Luka Milanovic, Dominique Zehnder

Abstract

For large pedestrian facilities a good understanding of macroscopic characteristics such as flow and mean speed is essential. Today, these parameters can be studied through various methods such as real pedestrian experiments and different model approaches such as the social forces model or AI-based simulations. In our work, we designed and built an agent based model using AI to simulate pedestrian motion in a plain. The model is built on a data-driven approach. Using our approach, we could successfully carry out agent based simulations to reproduce the training scenarios for the AI-model. In a next step, we carried out various experiments to assess the stability of our model and its ability to embody characteristics of real pedestrian movement.

Zurich
December 2020

Contents

1 Individual contributions

In the beginning of our working process, we all had very similar tasks, which was researching as much about pedestrian dynamics and neural networks as possible. During this time we already started writing some extra libraries for loading data (Dominique) and visualizing it (Michael/Alex). After this first phase we split up and Dominique and Michael continued working on feed forward networks and wrote the libraries we finally used, while Isabel and Luka researched and tried to implement LSTM networks and Alexander coded the visualization and presentation of our data. In the last phase of the project everyone worked on the presentation and report.

2 Introduction and Motivations

As the title implies, our project exists in the overlap of two interesting, but quite different, sciences - machine learning and pedestrian dynamics. Since machine learning using neural networks is a relatively new and quickly growing field, we thought that it would be nice to have at least some first hand experience with it. And because the context of our project is more about applying neural networks and less about working on the theory behind it, it was a good opportunity for us to gather our first experiences. Pedestrian dynamics on the other hand intrigued us mostly because the system itself, containing not just one, but many human minds, seems infinitely complex. But still it can be described with surprisingly simple models.

While searching for a more specific goal, we noticed one of the papers mentioned in the lecture about pedestrian dynamics [1]. It is about using neural networks to predict the speed of pedestrians. We wanted to take it a step further and also try to predict the velocity (speed and direction) of pedestrians.

This approach is very different to most historical approaches which view pedestrians as something akin to particles or fluids [2]. We believe that our more data driven approach could be used to view this problem from a different angle. A model more refined than ours could be used to test out different geometries for spaces where pedestrians should move in. This information could then be used to optimize those spaces and improve metrics, like maximum flow, dwelling time or to predict pedestrian trajectories in an autonomous vehicle. Results similar to this have recently been achieved with very involved model design [3].

3 Theory of Pedestrian Dynamics

There are two important relationships in the theory of pedestrian dynamics, average speed of pedestrians as a function of density and flow as a function of density. Flow is defined

as the average speed per area. A *Weidmann*-type model [4] predicts the movement in an open space follows the following relationship:

$$v(D) = v_0 \left(1 - e^{-\gamma\left(\frac{1}{D} - \frac{1}{D_{max}}\right)}\right) \tag{1}$$

With v as the speed at a certain density, v_0 the free-flow-speed of a pedestrian, γ a calibration constant, D the density, D_{max} the density at which no movement would be possible.

4 Description of the Model

Our model consist of multiple pedestrians moving in a continuous and infinite 2d plane. In each discrete time-step every agent moves to the position dictated by his own neural network. This means that this approach was agent-based, contrary to a model where one network computes the positions for all agents. Also in our model every pedestrian is controlled by a network of the same kind. To train this network we utilize data generated by real pedestrians.

4.1 Data

As training data we chose to use the data sets provided by the Forschungszentrum Jülich [5]. The data consists of top down videos of pedestrians moving in laboratory environments, consisting of different obstacles. In particular we focused on the data from the "Unidirectional flow, closed boundary condition" and "Bottleneck" data set. One main advantage of choosing this source for our data was that it was already processed. This means that the position of each person in the video in a fixed coordinate system was available as a text file. This allowed us to start training our networks faster and significantly reduced the time needed to process the data.

4.2 Model

A very simple model of a neural network can be seen in figure 1. Each neuron has n inputs and one output. In a standard neuron the output is:

$$y = ReLU\Big(\sum_{j=0}^{n} w_i x_i\Big) \tag{2}$$

$$ReLU(x) = max(0, x) \tag{3}$$

Where w_n are the weights for each input and ReLU the rectified linear unit function. ReLU can be exchanged with other nonlinear functions, like tanh, which is also often

used. A neural network consists of multiple such neurons which are interconnected and is a function from $\mathbb{R}^n \to \mathbb{R}^m$. The universal approximation theorem [6] states that a sufficiently big network, with the right weights, is able to approximate any function in that domain as infinitely close.

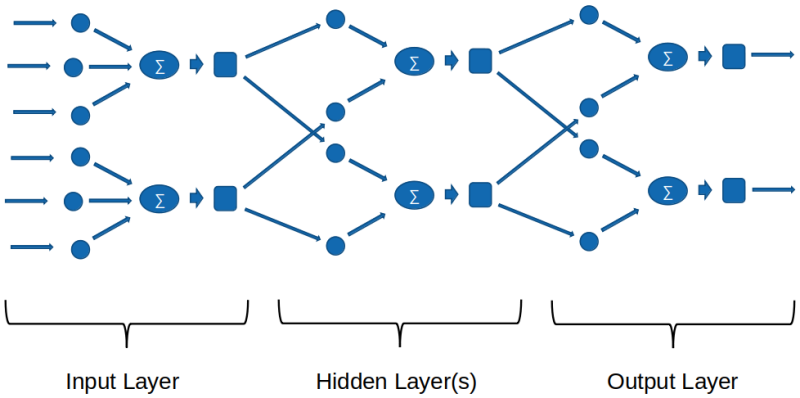


Figure 1: Graph of a simple feed forward neural network. The weights are represented as circles, the sum operation as ellipses and the nonlinearities as squares. The width of each layer is defined to be the number of neurons in it. In this case we have an input, an output and one hidden layer, all with a width of two (2,2,2).

Changing the weights until the network replicates the desired function is called training. Usually one does not know the exact function that needs to be replicated, just it's value at certain points. This means that the training data consists of pairs of input and output, also called truth value. In other words training the network means reducing the distance between output of the network and truth value at a finite amount of points. This distance is called loss. A very common method of doing this is called gradient descent. The weights are modified in a way that follows the gradient of the loss function, until a minimum is reached.

In particular most of the networks used by us were feed forward networks, meaning that each layer of neurons only fed into the next layer and there was no "feedback". These networks had topologies ranging from simple ones, like (28,50,20,50,4), to bigger ones used by other people for similar tasks [7].

4.3 LSTM

Another possible network topology would be a Long Short Term Memory (LSTM) network. Contrary to feed forward networks LSTM networks employ feedback, which means using the outputs of some neurons as inputs for neurons in earlier layers. This gives them the ability to save information for later use, which is important for analyzing sequences. This

architecture seemed to be quite popular in the field of pedestrian dynamics as indicated by the large amount of papers utilizing it [8, 3, 9].

The main difference in implementation is that LSTM networks are trained with sequences and not just independent input-truth value pairs. As an alternative to feed forward networks we have also tried implementing LSTM networks, but have not managed to achieve comparable results. This is why we decided to abandon LSTM networks for this project.

4.4 IO/Parameter

Before implementing the model, some decision had to be made about what kind of data to provide.

Using papers as a guideline, [1] helped us decide on some important factors. In their research, they used positions and velocities of surrounding neighbours and obstacles. The number of considered neighbours was varied in the different implementations.

Our conclusion was that the number of nearest neighbours was an important factor, as well as their positions, in addition to the position of the agent. Adding the velocities to the data could further improve the results.

As this would pose a challenge by itself, we did not consider the geometry of the situation, since the chosen data sets are fairly simple on that point; instead the neural network would learn it by the behaviour of the pedestrians.

As an output we expected the position (and velocity if wanted) of the agent in the next frame.

Parameter	Value	Meaning
neighbors	*6* / **7**	Number of neighbors to be considered by the agents in *corridor* and **bottleneck**
ret_vel	True	Enables adding velocity of the current position as an input
nn_vel	True	Consider velocity of the neighbours in addition to their position
truth_with_vel	True	Adds the velocity to the truth values
hidden_s	[32, 32, 64, 128, 128, 40]	6 hidden layers and the number of their units, adapted from [7]

Table 1: Setting of the parameters used during the simulations.

5 Implementation

We implemented our simulation in python using the PyTorch library. We chose python due to its platform independence and fast development time. With the use of dedicated and highly optimized libraries performance is no problem for the simulations. We chose PyTorch because of its clear and simple use of tensors and the provided basic functionality to implement neuronal networks. But any other machine learning framework would be usable. To further speed up development we used the jupyter notebook code environment that enables for more flexible development and advanced documentation.

5.1 Code structure

We identified the following code parts and organized the code in the same structure:

- Loading and preprocess data set
- Training the network
- Running simulations with agents
- Visualizing results
- Storing Models

Each individual part was implemented as Classes or functions in a separate module. From the main jupyter notebook we control the loading, training and simulations. Figure 2 gives an overview over the data flow through one experiment.

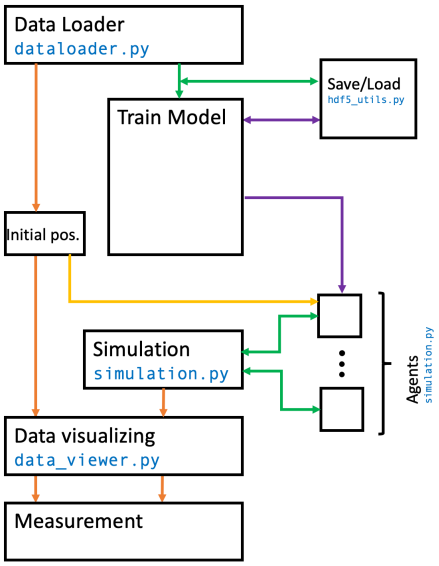


Figure 2: Data Flow; orange: DataLoader object, green: input data to network, purple: network weights, yellow: initial positions

data_loader.py

In this file we implement the DataLoader class which is used to extract the training data from the raw data set. Further we can use this object to store the trajectories of all people and agents in the data set and simulations. All further modules use this class as interface.

data_viewer.py

In this module a set of different plotting routines is implemented that can generate images and movies from a DataLoader structure.

simulation.py

In this part we implement an Agent class that is the smallest entity in a simulation. This agent accepts the surrounding neighbors as an input and will compute its next step. The agents are controlled by the Engine class which provides the agents with the input data and collects their output. In the end a DataLoader object is returned with the simulated agents.

hdf5_utils.py

For a good scientific method it is important to keep track of all parameters of an experiment. Therefore we implemented a custom save and load routine for training data and models. The advantage of the HDF5 file format is we can append metadata to the data and thus keep track of all parameters.

AgentTraining.ipynb

In this file we train the neuronal networks and conducted all experiments. We first implement the settings for the current training and experiment. Then we load the data with the given settings. We train networks or we load them from memory. Then we modify a DataLoader object in order to create the desired initial condition for a simulation. After a run through the simulation we select a homogeneous interval and measure mean speed and density.

5.2 Hyperparameter

The training data consists of the x and y coordinates of the agent in question and his n nearest neighbours, if desired the corresponding velocities can be used. Since only the trajectories and therefore the positions of the participants where included in the dataset, velocities have to be calculated. The velocities are calculated by taking the difference between coordinates of two consecutive frames and divided by the number of frames per second.

Downsampling was implemented by using the position and velocity of frame number (currentframe + downsample) as truth values.

The learning rate is initialized in the beginning of training, but will be further adapted during the process by multiplying it with a decay factor periodically.

For the values used in the resulting simulations, refer to Table 2.

5.3 Training

With the choice of our data set we also decided on training our model with supervised learning. The model is expected to output a prediction, which is compared to the target

Hypera-rameter	Value	Meaning
epochs	15	Number of loops where each crosses the training data set entirely
batch_size	10	Number of samples in one batch
downsam-ple	8	Predicting positions and velocities for the frame number (current frame + *downsample*)
lr	0.001	(initial) learning rate
decay	0.1	Factor applied to learning rate for decay
decay_step	5	Number of epochs before updating learning rate decay

Table 2: Setting of the hyperparameters used during the simulations.

or truth value, and based on that optimizations are made. This is different to unsupervised learning or reinforcement learning. In supervised learning, overfitting is a common phenomenon. We tried to counteract this by using a validation data set in addition to the usual training data set.

These data sets allowed us to evaluate the performance of the model. The validation data set is used to see if the model is overfitting on the training data. The validation set contains different trajectories from the ones in the training set. If the model performs well on the training data, but not on the validation set, it is a sign of possible overfit. This leads to a bad generalization ability of the model. It would not perform well on new, unseen data.

To be able to estimate the model's performance and accordingly adapt the weights and biases, we used the *mean squared error loss function*, which is the squared L^2-norm of the difference of the predicted value and the target value (our truth value).

Additionally to the loss function, an optimizer had to be chosen. In our case we decided to go with *ADAM*, a gradient-based optimization method commonly used in neural network applications.

6 Simulation Results and Discussion

With an operational framework to generate training data, train networks and carry out simulations we thought of ways to qualitatively and quantitatively analyze the behavior of our modelled pedestrians and compare it to the behavior of real pedestrian data. In order to manage our resources, we focused on the scenarios of a unidirectional corridor and a bottleneck. For both features the general walking direction is from left to right.

6.1 Reproducing Data

The first major result of our work was to train models that were able to reproduce the real-data behavior in close fashion. Fig. 3 shows the simulated trajectories and corresponding training data (raw data) for both corridor and bottleneck. The simulations were also visualized as videos that can be found on the GitHub page of this project.

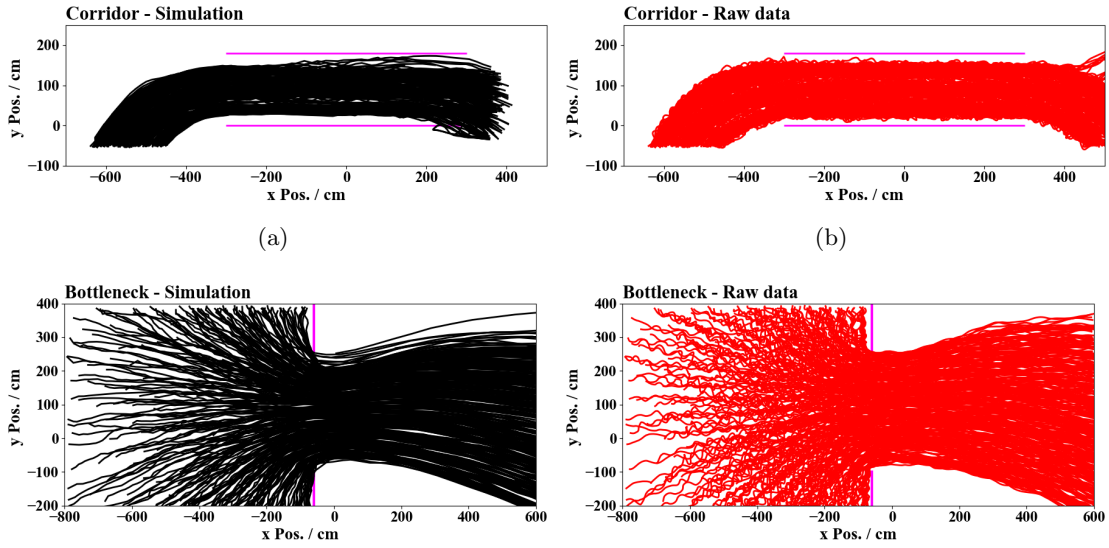


Figure 3: (a) and (b) show the real and simulated trajectories for a corridor setting, (c) and (d) show the respective trajectories for a bottleneck. Pink lines represent walls that were present in the real-pedestrian experiment.

6.2 Corridor

We used the corridor data-sets *ug-180-**** from Jülich university. The corridor in the experiments has a width of 1.8 m. If not stated differently, we used the *ug-180-095* data-set (95 people in the experiment) with the model-parameters stated in Table 1.

6.2.1 Perturbed Initial Conditions

We studied the impact of perturbation of the initial values on the simulation results. Fig. 4 shows the raw data and various simulation results for a time-span of 400 frames. The first simulation was carried out with the same initial conditions, for the second and third we added a random uniform perturbation of ±20 cm and ±50 cm respectively. The results show that for small perturbations the agents maintain a reasonable distance and behavior resembling the raw data, but for bigger perturbation the system diverges into a more

chaotic state, leading to even the wall geometry being neglected. Note that the latter can be directly explained by the lack of encoding of geometrical features as an input to our model. Also, the induced perturbation does not respect the wall geometry. Therefore, agents could be placed too close the the wall through the perturbation, which could also be a reason for the chaotic breakdown of certain agents.

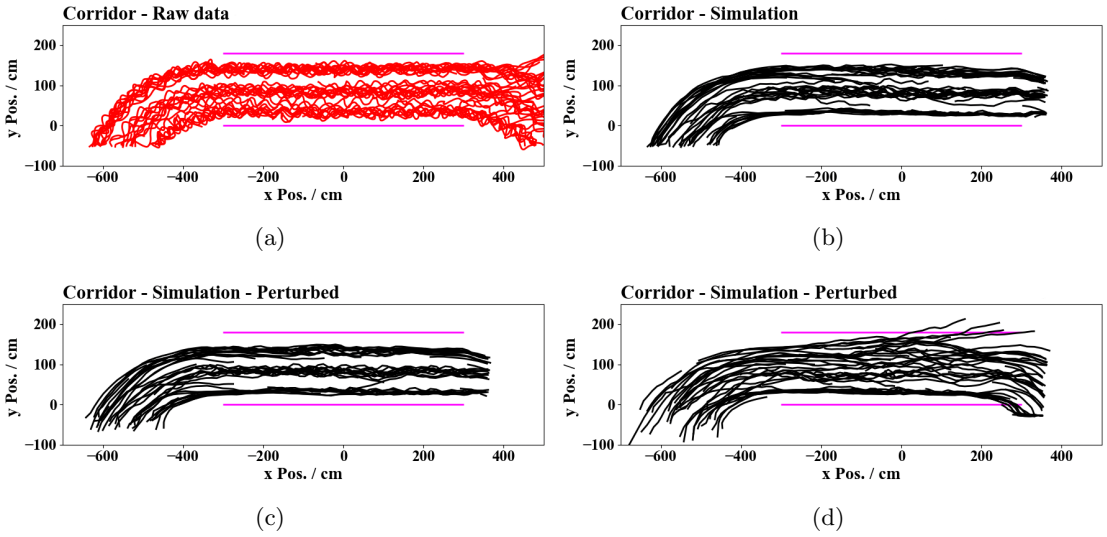
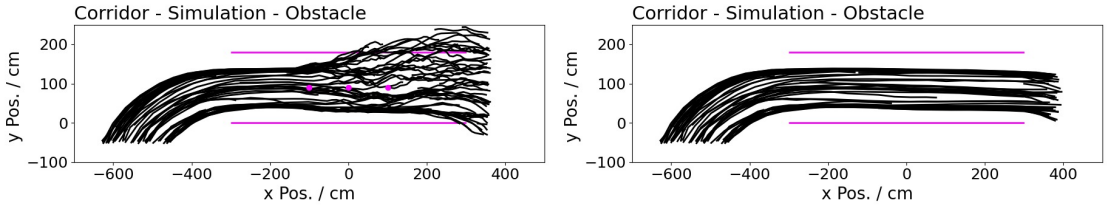


Figure 4: (a) Raw data. (b) Simulation with same initial conditions. (c), (d) Perturbed initial condition with a uniform perturbation of $\Delta = \pm 20\,\text{cm}$ / $\pm 50\,\text{cm}$.

6.2.2 Obstacles

Using additional static agents (fixed position in simulation), we tried to assess the reaction of our model to situations that were not present in training. Since the model "sees" its nearest neighbors for simulation, these static agents could in theory be used to build arbitrary geometries. Fig. 5 shows the trajectories of agents exposed to a geometric feature consisting of three static agents (in pink). The trajectories show that the agents indeed react to the new scenario and in many cases sidestep as it would be expected from humans. But since there is no hard-limits for the simulation agents, some also pass straight through the built wall.



(a) (b)

Figure 5: (a) Simulation where three "static agents" were placed in the middle of the corridor to resemble obstacles. The model was trained on samples of all data-sets *ug-180-****. (b) shows the reference simulation without the static agent for the same time-span.

The carried out qualitative measurements finally show that our model cannot only reproduce the exact initial data it has been trained on, but also react certain changes of the environment. This said, the model overall is still relatively unstable, leading to chaotic behavior of the initial conditions are changed too much. Also, the external geometry is only loosely adhered because it is not encoded as a model input. We think that an additional tracking of geometry and more refined training sets (more differing scenarios such as congestion behavior and human reaction to obstacles) would allow for models that resemble human behavior even better.

6.2.3 Adaptability

Moving away from qualitative observations, we tried to assess the adaptability of our model to different situations. For this, we trained a model on samples of corridor data with varying number of pedestrians (*ug-180-****) and used this model to simulate the pedestrians for all of the used data-sets (same initial conditions). From this we extracted the mean density and mean speed (euclidean) in a Region of Interest (ROI) for a time-span of 50 frames. Fig. 6 shows the measurement setup with the ROI. The number of frames was limited to 50 in order to prevent effects of congestion on the steady-state measurement.

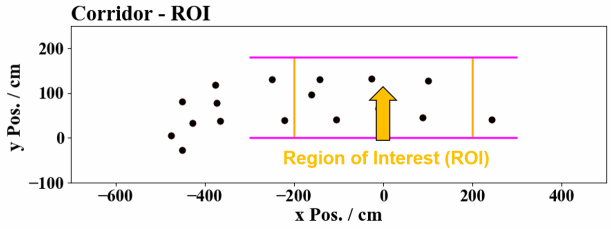
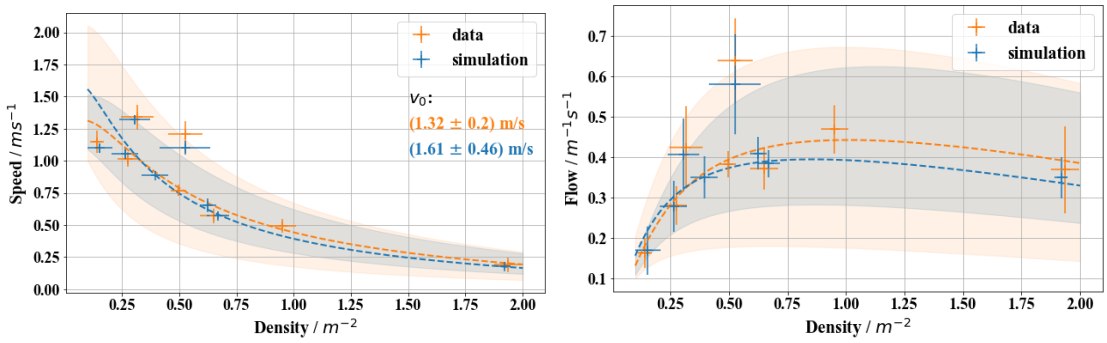


Figure 6: Simulation with agents (black dots) and Region of Interest (ROI) indicated with yellow lines.

In order to quantitatively interpret the measured speed and flow, we fitted a *Weidmann*-type model to the results. We used a maximum density of $D_{max} = 5.4\,\mathrm{m}^{-2}$ as proposed by Weidmann and fitted both the empirical parameter as well as the free flow speed v_0. Both v_0 and γ were fitted using orthogonal distance regression [10]. Fig. 7 shows the obtained results for the simulation and the raw data. The closeness of the simulation to the real data shows that our model is capable of adapting to various density scenarios and behaves in good proximity like the real pedestrians from the experiment.



(a) (b)

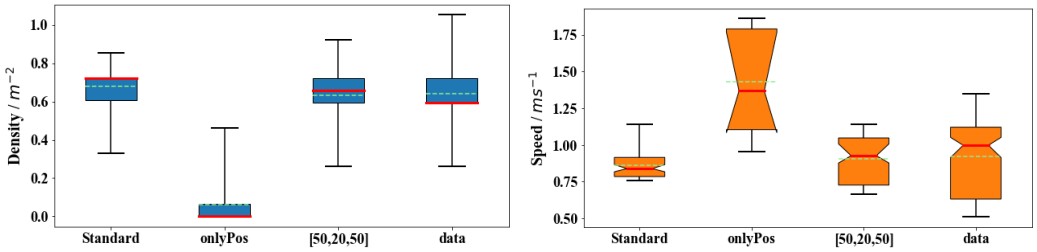
Figure 7: (a) Mean Speed and (b) mean Flow against density for various data-sets (ug-180-***) and corresponding simulation using the same model for all simulations. The errorbars represent a 1σ uncertainty around the mean values and the colored areas depict a 1σ uncertainty of the fit-parameters. Fitted γ values are 0.50 ± 0.15 and 0.34 ± 0.14 for data and simulation respectively.

6.2.4 Parameter Sweep

Lastly for the corridor, we tried to analyze the stability of the training process and find differences in simulation quality for varying training parameters. For this, we trained 10 models with the same parameters and measured the speed and density values in the ROI for a time-span of 500 frames. Qualitatively, all trained models behaved like the raw data, thus walking through the corridor with a reasonable speed and distance from each other. Numerically we found a mean speed of $(0.97 \pm 0.10)\,\mathrm{m/s}$ and a mean density of $(0.60 \pm 0.05)\,\mathrm{m}^{-2}$ (mean of the 10 simulations). The raw data obtains a mean speed of $0.90\,\mathrm{m/s}$ and mean density of $0.65\,\mathrm{m}^{-2}$. Therefore our model also numerically behaves similar to real pedestrians. During this analysis we noticed that the speed distribution for the simulations is less spread out than for the raw data. This indicates that our model behaves closer to the average of many pedestrians than to an actual microscopic pedestrian.

In a second measurement we varied the training parameters and observed the model performance (qualitatively and mean speed / density). The distribution of the measured

mean speed / density for different parameters is shown in Figure 8. We find that the model with only positional data (onlyPos) performs considerably worse than the others. From these simulations we also see that the smaller network ([50,20,50]) does not perform worse than the bigger one (see 1) considering mean speed and density, but has a data spread that resembles more the one of the original data. However, the small network has a qualitatively less stable behavior with individual agents sometimes performing very chaotic. We also want to note that simulations without *Downsampling* lead to unstable models that could drift into a chaotic behavior very rapidly.



(a) (b)

Figure 8: Distribution of the measured mean density (a) and mean speed (b) for different model parameters. The red line indicates the median, the green line the mean-value. The notches in the speed-plot indicate the 95 % confidence interval for the median. *Standard* is the parameter set mentioned in Table 1. For *onlyPos* only the agent and neighbor positions are given as an input.

6.3 Bottleneck

6.3.1 Training

For the bottleneck simulation we used the *ao-360-400* data-set (3.6 m corridor width / 400 people) as training set. As model parameters we chose 7 nearest neighbors.

6.3.2 Results

The more complex geometry of the bottleneck allows us to assess whether the trained agents are able to cope with more nuanced structures in our training data. This has on the whole been very successful.

Once the agents were trained on the *ao-360-400* data set we assessed their performance in three different environments. In setup I) we gave the agents the same initial positions as were the case in our training data, in setup II) we initialized them in a homogeneous grid on one side of the bottleneck and in setup III) we initialized each agent with a random initial position, again only on one side of the bottleneck. To asses the performance of the agents, both qualitatively and quantitatively, we again reverted to the concepts of density

and flow described above. In the following the density at a certain point (not necessarily the position of an agent) was calculated by counting the number of agents present in a radius of 70cm around that point. The flow was obtained by multiplying this with the average speed of those agents present. We will start with a few qualitative observations.

For each setup there were specific questions we wanted to answer about the behaviour of our agents. In setup I) we expected the agents to not imitate the training data precisely, while still showing similar and human behaviour. In Figure 9 we see the trajectories of the 'free' agents as opposed to the trajectories the real humans took in the training set. These are indeed not identical and can vary significantly for one person, but the agents still behave in a way that is realistic to the human observer. A notable effect we observe is the decreased waddling of the agents as compared to the humans. A possible reason is the downsampling or the lack of memory in the architecture of our model making it impossible for the agents to introduce an oscillation from one foot to another since it would only be a function of time rather than of the agents or their neighbor's position.

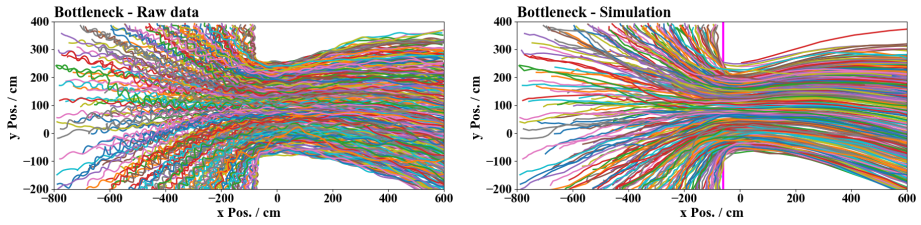


Figure 9: The agent simulated trajectories compared to training data with identical initial state.

With the first sanity check successful, we wanted to see whether the agent model generalizes to situations it has not seen before. In setup II we gave the agents random initial positions, resulting in local clusters with higher densities than could occur in the actual human scenarios. In the training data we observe an average density of 2 people/m^2 (excluding empty areas) we increased this to an average of 3.1 people/m^2. Fig 10 shows the random initial distribution of the pedestrian density for setup II. In setup III we gave the agents homogeneously distributed initial positions. To put the model further outside of its training regime we increased the mean density by 40%. This is not possible for realistic situations. The simulation result for both scenarios (II and III) were comparable to lower densities. This shows that our model performs robust even in situations it clearly has not been trained on. However, this behavior does not always resemble actual pedestrian behavior, since our model has no physical restrictions such as

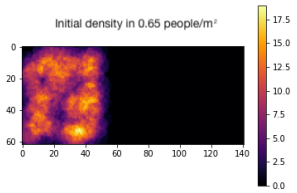


Figure 10: Initial density distribution in setup II

maximum pedestrian density. In Fig. 11 we can see that our simulation reaches pedestrian densities of over $17\,\mathrm{m}^{-2}$ which is approximately three times the density observable in a real situation.

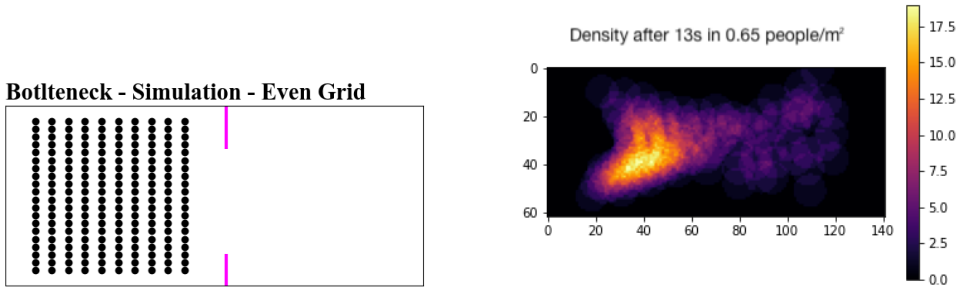


Figure 11: The symmetric initial grid positions of setup III and the density distribution after time evolution of 13 seconds.

Naturally it does not suffice to say that the behaviour of the agents looked adequate to the human eye. If our AI is to simulate real people it has to pick up on the more subtle patterns in our data, such as the flow charts discussed before. For our three setups we obtain the following charts:

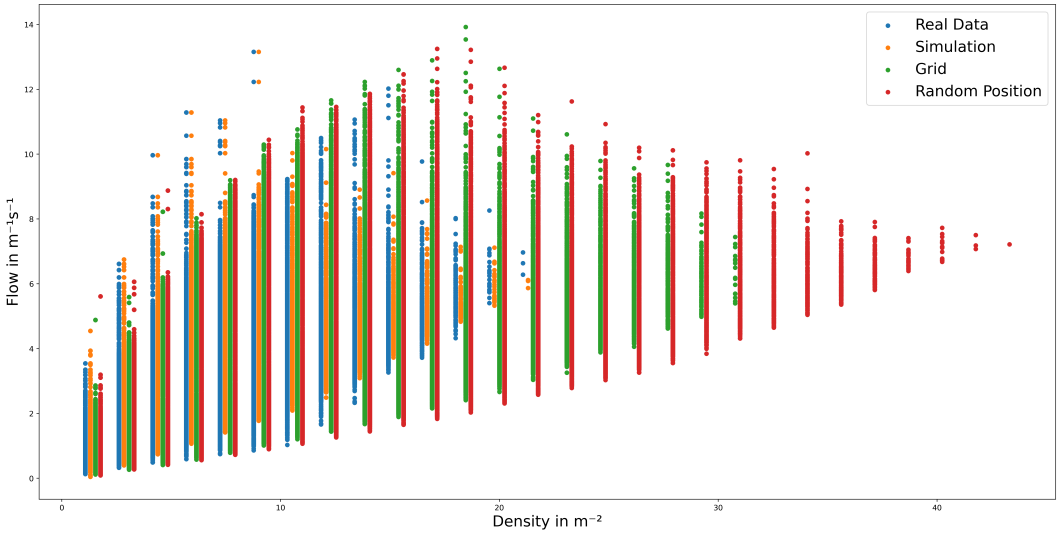


Figure 12: Flow charts for training data and three different setups.

We clearly see the same behaviour we observed for the corridor as well. It is evident that for setups II and III the flows are much higher than for the training data and setup I. Our model suggests that that agents are prepared to walk faster at the same congestion if

the overall density of the environment is larger. This finding is in accordance with similar studies on the topic and has already been established by [4]. We conclude that not only did the model learn to imitate the density dependence of the flow in pedestrian dynamics for the more complex setup of the bottleneck as well, but it also shows the ability to somewhat generalize this concept to situations it has not seen before, indicating that it might have learnt some deeper human aspects of pedestrian dynamics.

The main limitations of our approach are twofold and manifest themselves clearly in the bottleneck environment. While, as we can see in Figure 9, the agents do indeed respect the geometry of the setup they are not forced to do so. This means that small transgressions are possible and do indeed occur, making the simulation less reliable since they cannot be corrected for. The second limitation is that this effect becomes more dominant as the general density of the agents decreases. This can be seen in Figure 13 and limits the range of applications of our model as it cannot be used for low density problems.

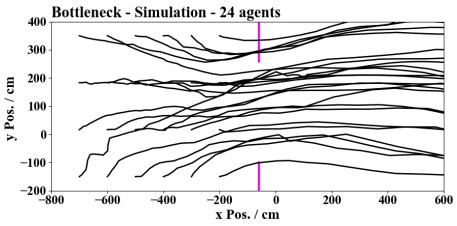


Figure 13: Agent trajectories for only 24 participating agents. Note the relatively frequent crossing of the bottleneck walls.

7 Summary and Outlook

In conclusion we have designed and trained a neural network which is able to imitate human behaviour in a pedestrian dynamics context. We have trained the AI for two different situations, a straight segment and a bottleneck. Density-Flow and Density-Velocity analysis shows that the agents behave in a way, similar to humans.

One big limitation, that could be addressed in future works, is that no spacial geometry, like walls, stairs etc., was given to the network as information. The AI extracted the features of the segment or bottleneck from the behaviour of the people in the training data. One way to explicitly encode the environment could be to use a sort of 'ray casting'. This could be implemented by returning the distance to the next wall in predefined directions.

Another point of research could be the analysis of different network topologies. Especially LSTM, which we have failed to implement successfully, seems like a promising contender. If one wants to avoid encoding the spacial geometry as an input to the network, we believe that adversarial neural networks could improve the ability of the agents to learn the geometry from their human counterparts.

Bibliography

[1] Antoine Tordeux, Mohcine Chraibi, Armin Seyfried, and Andreas Schadschneider. Prediction of pedestrian dynamics in complex architectures with artificial neural networks. *Journal of Intelligent Transportation Systems*, pages 1–13, 2020.

[2] Dirk Helbing and Péter Molnár. Social force model for pedestrian dynamics. *Physical Review E*, 51(5):4282–4286, May 1995.

[3] Mark Pfeiffer, Giuseppe Paolo, Hannes Sommer, Juan Nieto, Rol Siegwart, and Cesar Cadena. A data-driven model for interaction-aware pedestrian motion prediction in object cluttered environments. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[4] Ulrich Weidmann. Transporttechnik der fussgänger. transporttechnische eigenschaften des fussgängerverkehrs, literaturauswertung. Technical report, Zürich, 1992-01.

[5] Forschungszentrum Jülich. Data archive of experimental data from studies about pedestrian dynamics. `https://ped.fz-juelich.de/database/doku.php#crowds_in_front_of_bottlenecks_from_the_perspective_of_physics_and_social_psychology`.

[6] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[7] Mohammadhossein Bahari and Alexandre Alahi. Feed-forwards meet recurrent networks in vehicle trajectory prediction. Technical report, 2019.

[8] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 961–971, 2016.

[9] Bang Cheng, Xin Xu, Yujun Zeng, Junkai Ren, and Seul Jung. Pedestrian trajectory prediction via the social-grid lstm model. *The Journal of Engineering*, 2018(16):1468–1474, 2018.

[10] Paul T Boggs and Janet E Rogers. Orthogonal distance regression. 1990.