This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 899739

CROWD DNA

H2020 EU Fet-Open Project

Technologies for computer-assisted crowd management

www.crowddna.eu

Call: H2020-FETOPEN-2018-2019-2020-01

Type of action: RIA

Grant agreement: 899739

WP N°2:	**Modeling and Simulating High Density Crowds**
Deliverable N°2.1:	**Internal version of the simulator**
Task lead:	**URJC**
WP lead:	**UL**
Version N°:	**1.0**
Date:	**29/04/2023**

Document information	
Deliverable Nº and title:	**D2.1 – Internal Version of the Simulator**
Version Nº:	**1.0**
Task lead:	**URJC**
WP lead:	**URJC**
Author(s):	**Dan CASAS (URJC), Alexis JENSEN (Inria), He WANG (UL), David WOLINSKI (ONH)**
Reviewers:	**Julien PETTRE (Inria), Solenne FORTUN (Inria)**
Submission date:	**29/04/2023**
Due date:	**30/04/2023**
Type:	**DEM**
Dissemination level:	**PU**

Document history			
Date	**Version**	**Author(s)**	**Comments**
12/01/2023	**0.1**	**Solenne FORTUN (INRIA)**	**Template creation**
22/04/2023	**0.2**	**Dan CASAS (URJC), Alexis JENSEN (INRIA)**	**First draft of Sections 1.1, 1.2, and 1.6.**
25/04/2023	**0.2.1**	**He WANG (UL), Jiangbei YUE (UL)**	**First draft of Section 1.3 and 1.4**
26/04/2023	**0.7**	**David WOLINSKI (ONHYS)**	**Section 1.5**
26/04/2023	**0.8**	**Dan CASAS (URJC)**	**Executive summary and all sections integrated**
27/04/2023	**0.9**	**Julien PETTRE (INRIA), Dan CASAS (FZJ)**	**Internal review**
28/04/2023	**1.0**	**Solenne FORTUN (INRIA)**	**Final layout**

Table of Contents

Table of Figures

Acronyms and Abbreviation

CDI	Crowd Dynamics International Limited
EC	European Commission
EMT	Executive Management team
FZJ	Forschungszentrum Julich Gmbh
GA	Grant Agreement
INRIA	Institut National De Recherche En Informatique Et Automatique
KPIs	Key Performance Indicators
ONH	Onhys
PO	Project Officer
UL	University of Leeds
ULM	Universität Ulm
URJC	Universidad Rey Juan Carlos
WP	Work-package

Executive Summary

This deliverable describes the efforts done during periods 1 and 2 in the Work Package 2 of the CrowdDNA project towards developing a new crowd simulator algorithm tailored to model both macro and micro-level crowd characteristics. As a reminder, the overall objective of WP2 is to *deliver a new generation of crowd simulation techniques that can predict crowd behaviors at macroscopic scales from numerical models of physical interactions*.

To this end, the consortium has worked on several key ingredients for a new crowd simulator that will be used to model macro- and micro-level interactions. In particular, we have developed methods to create 3D synthetic crowds from 2D trajectories and integrated several solutions to model and resolve the contact between individuals. Additionally, we have also improved the macroscopic behavior of the crowd by learning a novel neural social physics model that considers the influence of the environment in the crowd behavior.

Figure 1 depicts the different components of the proposed simulator, next to the CrowdDNA partner who led each of the developments.

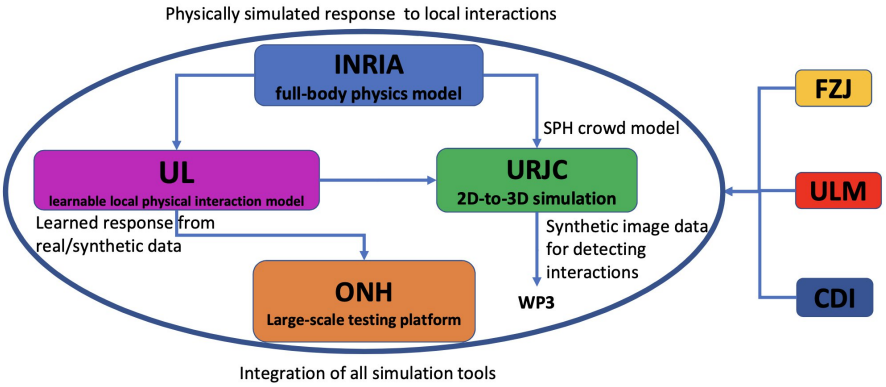

Figure 1. Components of CrowdDNA simulator.

All in all, our proposed crowd simulation agglutinates several key properties to improve existing over-simplified solutions for crowd simulation that are unable to model micro-level features. The CrowdDNA consortium will leverage the crowd simulator described in this document to achieve the overall goal of the project: the development of tools for a novel of crowd motion analysis.

1. Crowd Simulator

1.1. 3D Articulated Humans from 2D Disks

To tackle the grand challenge of analyzing the crowd behavior at micro-scale level, an important goal of the project is to extend classic 2D disk-based crowd simulators to generate simulations of crowds of articulated 3D humans. The key underlaying idea behind this objective is to use these 3D crowds, and their subsequent micro-level information (*e.g.,* contact forces between individuals, 3D body pose information, joint positions, limb velocities, etc.), as ground truth to train machine learning algorithms to infer microscopic information from videos of dense crowds.

To this end, as a starting point, we use the state-of-the-art UMANS (van Toll 2020) model to generate 2D trajectories of human crowds. UMANS is a solution that implements many crowd simulation algorithms, and it allows to easily configure new scenes and parameterize initial conditions of the crowd such as number of agents or location of the agents, as well as different properties of their macroscopic behavior. We exhaustively tested out the capabilities of UMANS and looked for the optimal method and parameterization for the purposes of CrowdDNA project.

While evaluating current 2D crowd simulation methods, we considered multiple aspects of the agents that affect the realism of the output 3D crowd, such as distance between agents, distance to the obstacles, trajectories, and collision-avoidance behaviors. We investigated the methods of Karamouza *et al.* (Karamouzas y Overmars 2010), Moussaid *et al.*, PLE (Stephen J. Guy 2010), and RVO (Jur P. van den Berg 2008), and concluded that a key parameter for our interest are the "range parameter" and "time to collision", which control the distance from which agents react before colliding with and obstacle or other agent. Unfortunately, we realize that no policy in the existing methods was able to avoid collisions in a natural way: if time to collision was increased, agents spread more, leading to highly unrealistic dense crowds when visualized as 3D articulated agents; if time to collision was reduced, agents get easily stuck into obstacles.

Our solution is to extend the RVO (Jur P. van den Berg 2008), algorithm and separate "time to collision" into "time to collision to agents" and "time to collision to obstacles". This allows the modeling of dense crowds, by setting the "time to collision to agents" very low, while avoiding collisions and getting stuck to obstacles, by setting the "time to collision to obstacles" high. This mixture of behaviors enables RVO algorithm to generate 2D trajectories of dense crowds in complex scenarios that resemble the macroscopic behavior of a real human crowd.

This allowed us to generate a large collection of 2D crowds in various scenes that present some potentially dangerous situations such as bottleneck scenarios, obstacles, and highly dense areas. Figure 2 depicts two frames of two different sequences generated with UMANS using our enhanced version of the RVO algorithm.

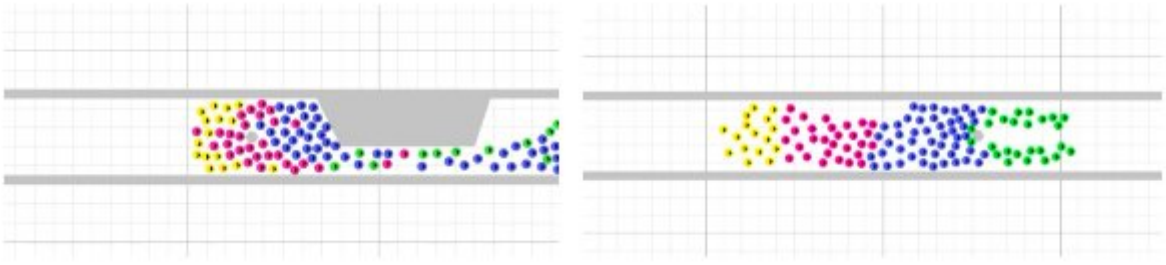

Figure 2. Two representative frames of 2D simulations obtained using the tool UMANS. We generated a large collection of 2D simulations of crowds in scenarios that can potentially lead to dangerous situations, such as bottleneck corridor or obstacles.

Leveraging these 2D crowd simulations, our next goal was to generate 3D photorealistic crowds. The key underlaying idea is to animate 3D characters such that they follow the 2D trajectories computed with the 2D crowd simulator. This requires synthesizing articulated 3D humans that move in a natural way while following the computes trajectories. To ensure a realistic output, we need to guarantee that the locomotion and appearance of the 3D humans is natural.

To the end, we built a novel Unity solution that is able to import UMANS generated 2D crowd trajectories and generate the corresponding 3D crowd animation. As a basic 3D human, we use the popular model SMPL (Loper 2015) which provides a parametric model for humans. SMPL is controlled by both shape and pose parameters, which allows to synthesize any human any pose. To fulfill such 2D to 3D task, we first have to figure out what information of the 3D crowd was important to export from UMANS, and how to import it into Unity. To encapsulate all this information, we created the file format depicted in Figure 3, which consists in the (x,y) position (2nd and 3rd columns) of an agent at each time step (1st column). Additionally, we also export information about time to collision to other agents and obstacles (last two columns).

```
0.1,  -10.6775488,  -5.1308928,  1.0,  -3.52209e-08,  0.613845
0.2,  -10.6627175,  -5.1300966,  1.0,  -5.90995e-08,  0.431435
0.3,  -10.6471052,  -5.1292313,  1.0,  -7.59911e-08,  0.528974
0.4,  -10.6307963,  -5.1282872,  1.0,  -8.83025e-08,  0.479376
0.5,  -10.6138438,  -5.1272533,  1.0,  -9.74723e-08,  0.431041
```

Figure 3. Example of 2D information extracted from UMANS.

Using the file format from Figure 3, we are able to load 2D trajectories on a Unity solution. The remaining challenge to be solved is how to animate a SMPL (Loper 2015) human body model such that it follows the path while performing a realistic animation. We propose to use the Unity built-in solution Blend Tree, depicted in Figure 4, which is typically used to map user input (e.g., keyboard or game pad controller) to character motion, for example, when the user press the up bottom the character jumps while walking.

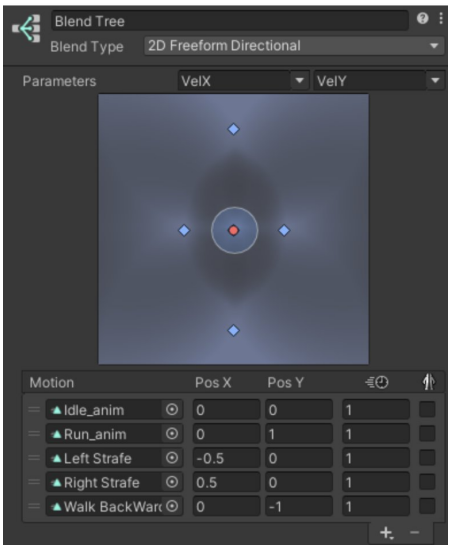

Figure 4. Unity Blend Tree used to convert 2D trajectories to 3D articulated poses.

Our idea is to make a slightly different use of Blend Tree functionality and use it to map 2D trajectories to 3D motions. More specifically, we pick 5 motions from a motion repository (idle, walk forward, run forward, turn right, turn left) and set up a Blend Tree that maps 2D velocities to a mixture of these motions. To compute 2D velocities, we simply apply central finite differences to the 2D position at times t-1 and t+1. In Figure 5 we visualize a set of 2D trajectories of a bottleneck scene loaded into our Unity 3D project, each trajectory colored using a distinct color, and the corresponding 3D humans walking along such paths.

Figure 5. A 3D crowd passing through a bottleneck scenario. Notice the 2D trajectories, originally computed using a 2D crowd simulator, depicted as color lines.

When using Blend Trees to lift 2D trajectories to 3D humans, we have to pay special attention to two sources of potential issues: foot sliding, which is caused when mapping a locomotion animation to a faster velocity; and inter-character collisions, which is caused by locating the 2D disks to close to each other.

We address the first challenge by computing the velocity of the hip in the base waking motion that we used in our Blend Tree, and then interpolating the corresponding motion such that it matches the target velocity of our input UMANS file. This ensures that the displacement produced by the locomotion cycle of the basic motion matches the target displacement in the 2D trajectory at this particular time stamp.

To address the second challenge, inter-character collisions, we run an exhaustive search of the optimal "disk size" in the 2D crowd simulator such that the subsequent 3D humans do not interpenetrate. However, we could not find an ideal configuration: if the disk size is set too big, the distance between 3D characters is too large to perceive the crowd as "dense crowd"; if the disk size is set too small, the 3D characters interpenetrate each other in dense situations. Hence, we opt for a different approach and propose to solve the inter-human collisions using a second step, described in Section 1.2, where we described a physics-based solution to resolve the collisions.

All in all, at the end of this step we have successfully generated animations of 3D crowds from 2D disk trajectories. This is depicted in Figure 6, where a side-by-side of a 2D crowd and the corresponding 3D crowd is visualized. Next, we discuss how we resolved the residual collisions between the 3D characters.

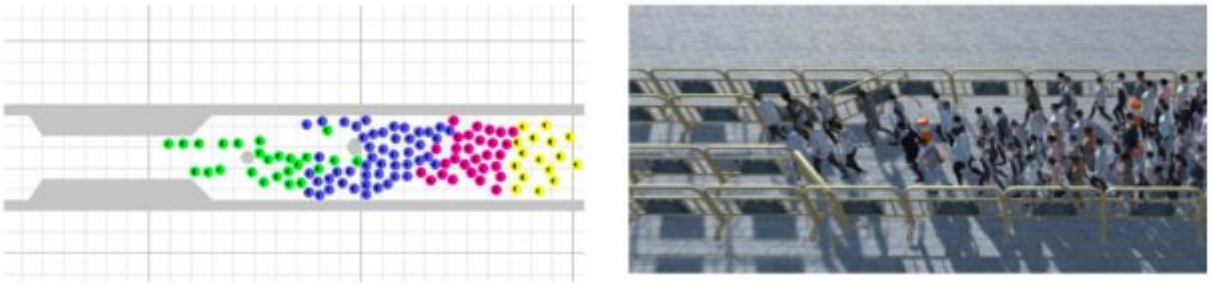

Figure 6. 2D crowd simulation from UMANS (van Toll 2020), representing a bottleneck scenario (left). Corresponding photorealistic 3D animation (right).

1.2. Collision-Aware 3D Human Crowds

To synthesize realistic 3D crowds, we do not only need to model the macroscopic behavior (*i.e.*, the overall motion of the crowd), but also a pay attention to microscopic actions between individuals that are in close contact. More specifically, we are interested in adding collision awareness at the individual level of the simulation, enforcing that the 3D body of each articulated character does not intersect with others.

Collision detection and handling is generally a hard problem in Computer Graphics, and robust solutions for deformable objects usually require expensive runtime computational costs, which does not scale well in scenes of dozens of individuals, each of them represented as a mesh with thousands of vertices. Hence, in order to provide collision-awareness to our 3D crowd simulator, we opt for approximating the character volume with a set of coarse volumetric proxies which can be used to very efficiently detect collision with other points proxies in the scenes.

Our formulation works as follows: we first approximate each character volume with a single cuboid, and check if it is overlapping with the other cuboids of the scene using the well-known axis-aligned bounding boxes (AABB) algorithm. For those pairs of cuboids that are in contact, we check for collisions for each pair of their corresponding volumetric proxies. Since these proxies are modelled with an analytic expression, checking for proxy-to-proxy intersections is a very fast computation.

To resolve the detected collisions, we formulate an articulated rigid-body simulation using as a soft constraint the underlaying locomotion and as a hard constraint the restrictions due to collisions detected initially. We solve the resulting system of system of equations using Newton methods, leading to a set of pose parameter per character that fulfill the contact constraints while maintaining the overall motion of the crowd.

Figure 7 presents a visualization of 3 articulated characters and their corresponding volumetric proxies. Figure 8 showcases a 3D crowd animation going through a bottleneck scenario, before and after resolving the collisions. As it can be seen, without introducing our formulation for collision awareness, the agents suffer from unrealistic intersections (notice how the zoom-in characters overlap each other). After adding collision awareness, the geometry of the characters does not overlap, producing realistic crowd effects such as pushing.

Figure 7. Visualization of the collision proxies used to detect intersections between 3D humans.

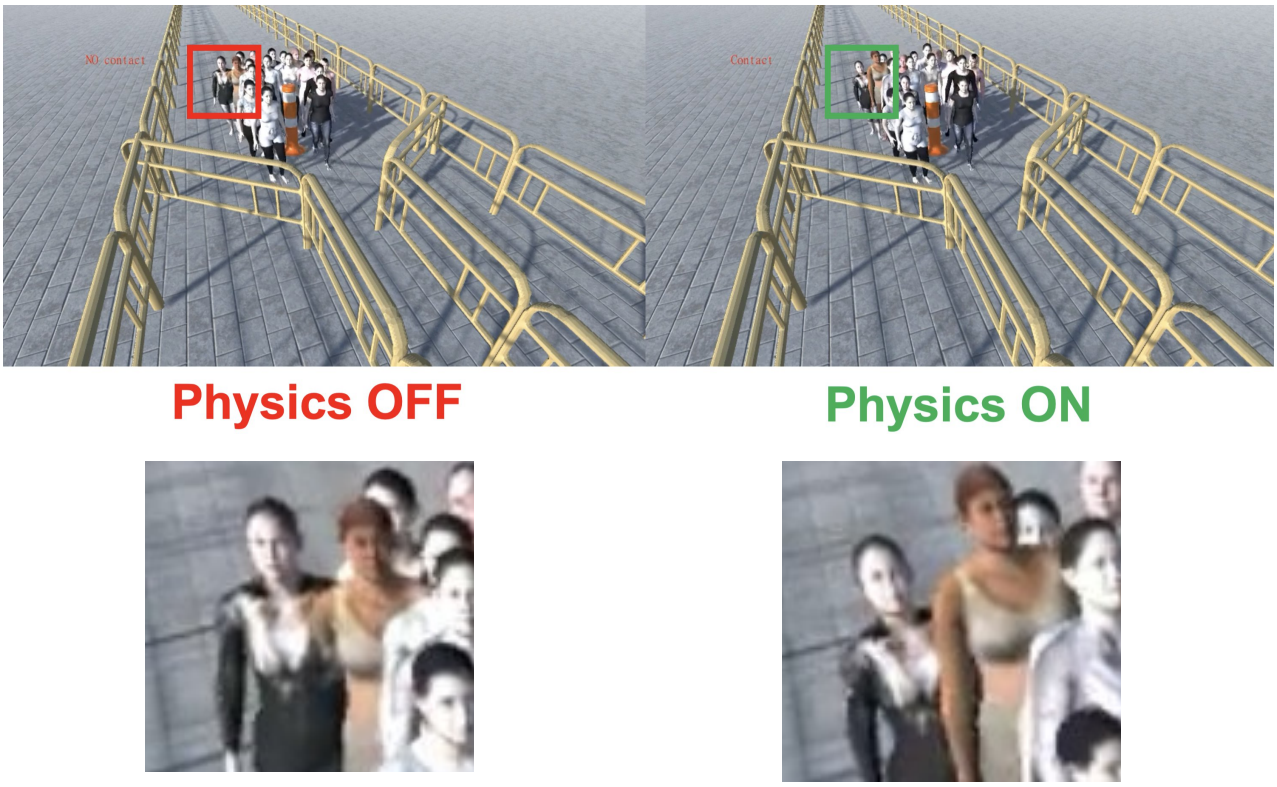

Figure 8. Comparison before and after solving inter-character collisions (top). Close-up of collision between two characters (bottom-left), and the result after resolving the contact (bottom-right)

1.3. Neural Social Physics

One key topic in crowd simulation is the realism of pedestrian behaviors. Achieving high fidelity in low-level behaviors such as steering is crucial for CrowdDNA as well as in general crowd research. Existing approaches generally fall into model-based and model-free methods. Early model-based methods tend to be empirical or rule-based methods derived via the first-principles approach: summarizing observations into rules and deterministic systems based on fundamental assumptions on human motion. In such a perspective, social interactions can be modeled as forces in a particle system or an optimization problem, and individuals can be influenced by affective states. Later, data-driven model-based methods were introduced, in which the model behavior is still dominated by the assumptions on the dynamics, e.g. a linear dynamical system, but retains sufficient flexibility so that the model can be adjusted to fit observations. More recently, model-free methods based on deep learning have also been explored, and these demonstrate surprising trajectory prediction capability.

Empirical or rule-based methods possess good explainability because they are formed as explicit geometric optimization or ordinary/partial differentiable equations where specific terms correspond to certain behaviors. Therefore, they have been used for not only prediction but also analysis and simulation. However, they are less effective in data fitting with respect to noise and are therefore unable to predict accurately, even when the model is calibrated on data. Data-driven model-based methods (e.g., statistical machine learning) improve the ability of data fitting but are restricted by the specific statistical models employed which have limited capacities to learn from large amounts of data. Finally, deep learning approaches excel at data fitting. They can learn from large datasets, but lack explainability and therefore have been mainly used for prediction rather than analysis and simulation.

Jiangbei Yue and He Wang proposed a new method combining the model-based and model-free approaches to learn the motion of pedestrians and the influence of the environment (*e.g.* obstacles) onto individuals, which is crucial in crowd management. In this research direction, the main aim is to learn explainable pedestrian behaviors from data. As a result, we proposed two novel human trajectory prediction methods considering the environment and finished two papers.

The first paper is “Human Trajectory Prediction via Neural Social Physics”, where we proposed the Neural Social Physics (NSP) model. This paper has been published at The European Conference on Computer Vision (ECCV 2022). NSP can explain pedestrian behaviors and retain good data-fitting capabilities by incorporating explicit social force models and deep learning. Based on exhaustive evaluation, Our NSP outperforms the state-of-the-art methods in standard trajectory prediction tasks on many public datasets and metrics. In addition, NSP possesses the ability to generalize to unseen scenarios with higher densities where NSP can predict more plausible motions than pure black-box deep learning methods. Our NSP not only performs well in prediction accuracy, but also gives explanations of corresponding predictions, which is key to understanding human trajectories, shown in the Figure 9.

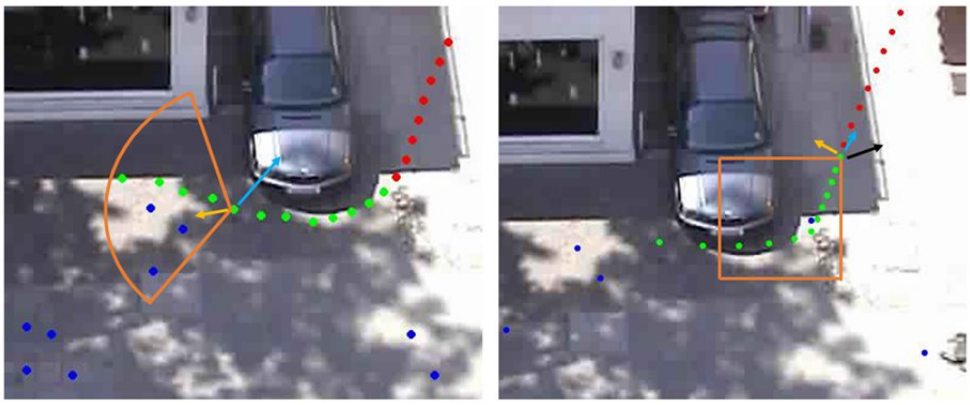

Figure 9. Two examples of NSP predictions. The future trajectories (green dots) are predicted given the past trajectories (red dots), where the NSP model predicts three kinds of social forces at each step to calculate the next position. We use yellow arrows, light blue arrows and black arrows to denote the goal attraction force, collision avoidance force and environment force, respectively. Orange areas are the view fields used in the calculation of the collision avoidance force and environment force.

In Figure 9, NSP can also provide plausible explanations of the predicted motion, by estimating the ‘forces’ exerted on a specific person. Figure 9 (a) shows that a person moves upwards instead of moving to his goal directly. This can be explained by the influence of the goal attraction force and the collision avoidance force ensuring that the agent can avoid other pedestrians (blue dots). Similar explanations can be seen in Figure 9 (b). The goal attraction force drives the agent to his goal, while other two forces make other pedestrians and the environment repel the agent.

The second paper is “Human Trajectory Forecasting with Explainable Behavioral Uncertainty”, where we proposed the Bayesian Neural Social Physics (BNSP) model explicitly considering explainable aleatoric and epistemic uncertainty. This paper is the extension of NSP and is going to be submitted to a journal. While enjoying all the advantages of NSP, e.g. high prediction accuracy, explainability of behaviours, etc., BNSP further enhances NSP by providing a confidence analysis for the prediction and explanation.

Figure 10. Based on the same visualisation scheme as NSP, we show the same predictions, but with confidence maps about the prediction and behaviours, shown as heatmap.

Figure 10 shows the predictions for the same two people in Figure 9. BNSP predicts the distributions (shown as heatmap) of these forces each step to calculate the next position. These heatmaps show the confidence of the estimated social forces in space and time, which can be also interpreted as the behavioral uncertainty of the pedestrians. BNSP possesses better explainability that NSP by further analysing the indeterminacy of the future trajectories. We can use the additional information brought by BNSP on confidence of these social forces to understand and analyse the motion of crowds, which is crucial in crowd management.

1.4. Learn to Predict Individual Reaction to Physical Interactions

Physical interactions in high-density crowds are ubiquitous. Being able to accurately predict reactions to physical interactions such as pushing and nudging is vital for assessing any potential physical danger, e.g. possible falling of individuals or collapse of people in crowds.

Based on the data capture in WP1, we learn realistic reaction motions via a new class of scalable models that combines an explicit physics model with deep learning. While the physics model, which is an Inverted Pendulum Model (IPM), is fast and scalable to capture the general motion trend, the deep learning component provides strong learning capacity to predict full-body motions.

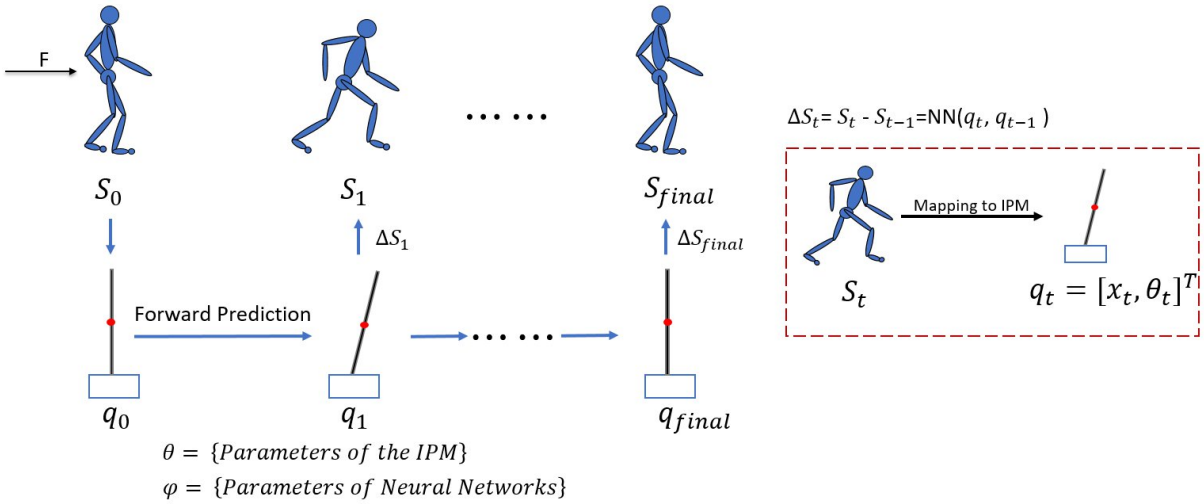

Figure 11. Overview of the individual-level model.

We show the overview of the model on the individual level in Figure 11. Given the full-body pose of the start frame S_0 and the external force F, we map the full-body pose to the IPM-level pose and make the forward prediction on the IPM level through the IPM engine and the predicted generalized force from neural networks. Then, we predict $\Delta S_t = S_t - S_{t-1}$ through neural networks from current IPM state and last IPM state (q_t and q_{t-1}) to

compute the full-body pose at each step. Finally, we optimize all parameters (θ and φ) via our loss function between the predicted motion and their ground truth.

We realised the IPM engine in 2D and 3D by deriving the motion equation from Euler-Lagrange equations to model the forward motion simulation. We built a visualisation tool for the IPM motion for evaluation. In addition, the 3D IPM engine has the ability to map the full-body pose to the corresponding IPM state. Based on the 3D IPM engine, we realize the individual-level model by exploiting a recurrent neural network to predict the generalized forces and anther network to predict the full-body poses.

Currently, we pre-processed raw data captured by FZJ for training and testing our model. The raw data include excessively redundant information and need to manually cropped and labeled. We first visually identify and crop the motion segment that is mostly relevant to physical interaction between people. We define the start frame as the first frame being exerted forces and define the end frame as the frame where the agent recovers the balance. The refined raw data are then fed into a IK-solver to output the BVH files to generate high-quality smoothed motions.

For training, we have designed single-agent and across-agent regimes, where the former trains the model on the initial period of time of a motion and the let the model to predict the rest. After obtaining initial success, we extended the training to across-agents, i.e. non-overlapped data in training/testing from different people, for better model generalization. Now, we have obtained good numerical and visualization results on the IPM-level. We show some qualitative results below.

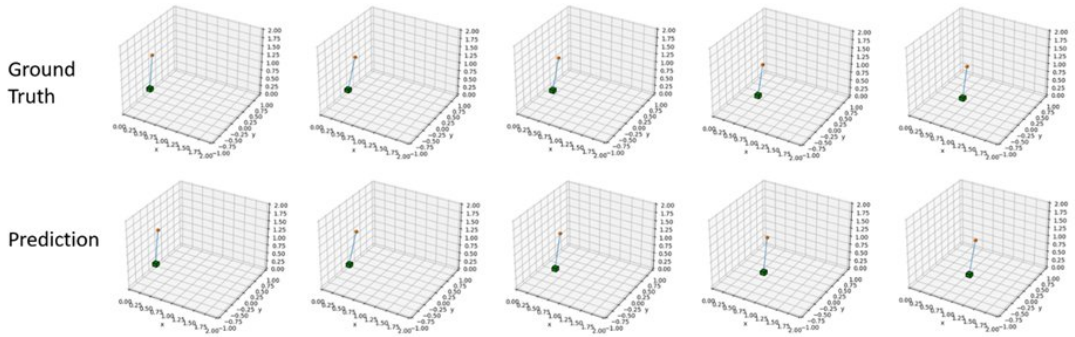

Figure 12. One example of prediction on the IPM level. Our method can accurately predict the motion as well as the IPM parameters such as the rod length.

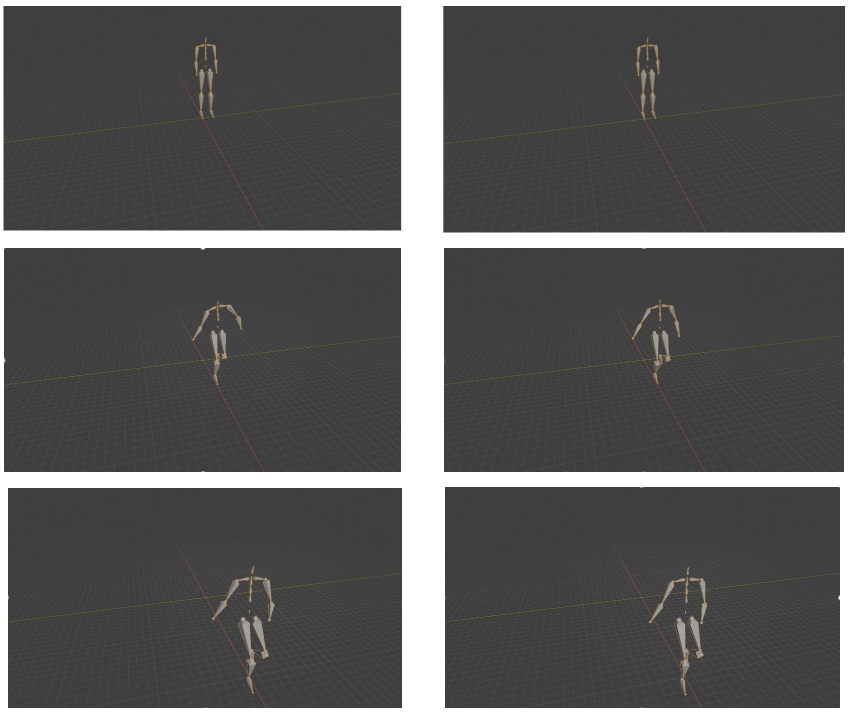

Figure 13. One example of prediction on the fully-body level. Left: ground-truth. Right: Prediction.

1.5. Microscopic Crowd Behavior

Tests, demos, and validation efforts will be difficult to showcase on simplistic scenarios. By simplistic scenarios, we mean crowds of people simply standing in an open environment. The issue will be that such scenarios, while possibly adequate for basic shockwave propagation demonstrations, will be very difficult to compare against actual crowds, moving and gathering within, for instance, a festival.

The festival example is here quite important, as thus far, the data that was successfully captured as part of the project only concerns this type of venue. Festivals, which are therefore our current target demonstration environment, present two main properties: (1) they often consist of more than one performance stage and (2) the geometry of the stages themselves can vary greatly. As a consequence of these properties, a user intent on configuring such events will face the major issue of determining the simulated agents' destinations. This is something that is entirely absent in an open environment, where the user can simply "pop" agents somewhere in the middle, and then direct them towards a common goal in order to build up density. In the case of a festival on the other hand, agents need to correctly spread around a stage, independently of its geometry.

This task of gathering around a stage is, in fact, not only harder than it appears, but also necessarily performed automatically. The obvious, low effort means of achieving it, is to "pop" the agents in roughly equal amounts around the stage, and then direct them to converge towards said stage, closest-distance style. It could be discussed here, if such a method does or does not inject bias as to the final spatial distribution of the agents. But the main problem that can be observed here, is that such a configuration method breaks the moment these agents are required to transition to another stage. Indeed, if agents are made to move to another stage, it will be impossible to apply the same configuration strategy there, given that the agents already exist, and that they are all in the same general location with respect to the second stage.

Unfortunately, this phenomenon of inter-stage movement is quite prevalent, as often festivals establish performance schedules in such a way that the audience is expected to attend a performance in one location, while another location is being prepped between performers. Such a rolling schedule therefore allows for smooth and wait-less transitions between performances. Such was the case at the Hellfest, where such an inter-stage movement was captured (see WP4). Given the constraints imposed on such a crowd movement (i.e. people already part of a dense crowd, moving towards another area which might require navigating around stage-delimiting barriers), we expect such a phenomenon to be difficult, possibly dangerous, and therefore necessarily present in the validation simulations.

Given what we have already established about the impossibility to control, in the general case, the initial location of a crowd before its movement towards the stage, we have updated the decision process of ONH's simulated agents to automatically decide what their destination will be in such a situation, irrespectively of their origin.

As a result of this effort (see Figure 14), our agents are now able to process the environment around a stage in such a way as to estimate for each possible direction (1) where they are likely to end up if they follow this direction (end position) and how close it will be to the stage (motivation), (2) how long they will have to travel in order to reach this end position (locomotion effort), and (3) how difficult it will be to navigate to this end position (navigation effort among the flow of other, surrounding agents). Additionally, note that all these estimations are made in anticipation, and therefore concern a future state of the crowd, made probable by its current overall movement. Finally, this choice is never final and is continuously updated as the agents move around the stage. As a result, our agents are now able to automatically spread around a stage, and their choice can further be controlled by the user through manipulation of the cost function's parameters.

This last aspect will require careful calibration following data collected at the various Crowd Observatories.

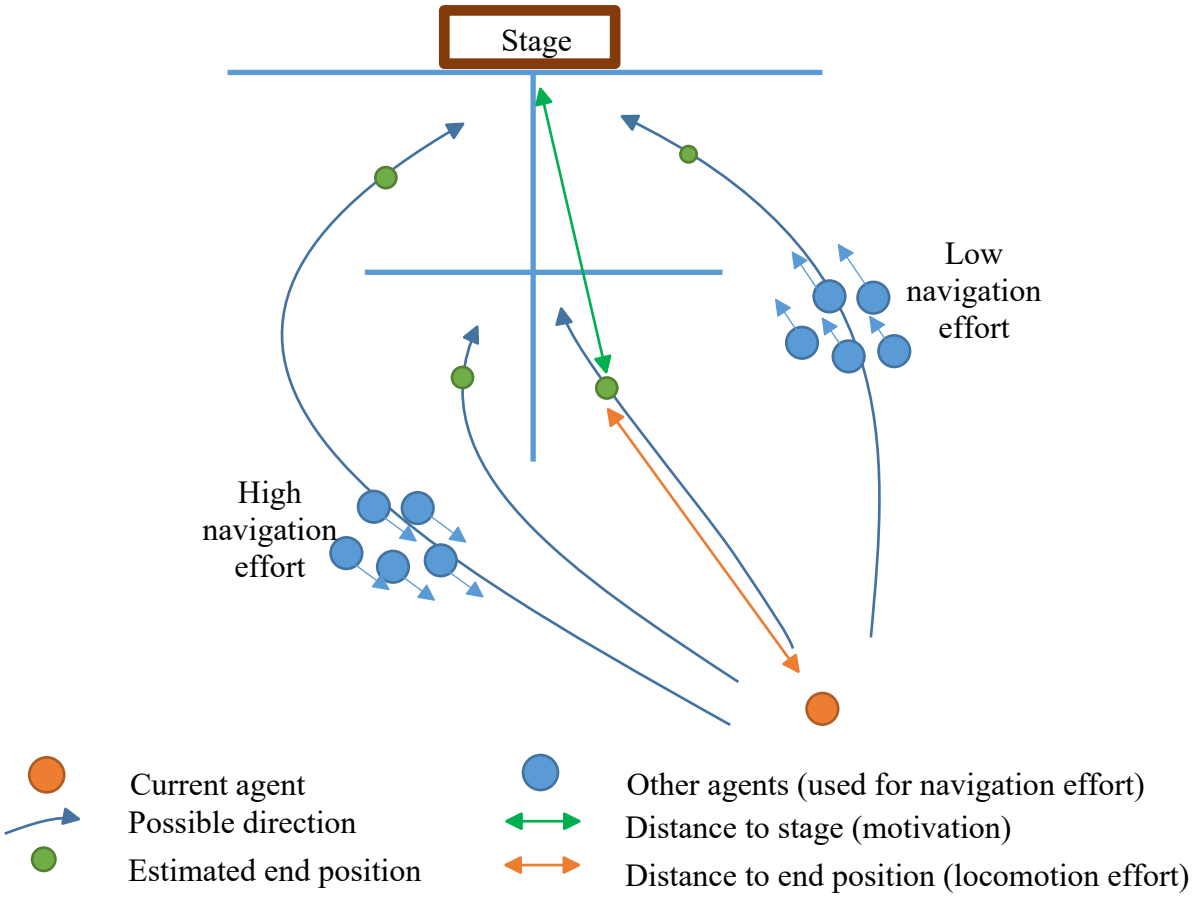

Figure 14. Quantities used for an agent's decision process when deciding where to go around a stage.

Figure 14 shows a still from a simulation of a cross-shaped stage positioned at the Main Stage area of the Hellfest (although several kinds of temporary stages have been used there over the years, this particular stage was not in place at that location during the 2022 edition, we simply reused the 3D model of this festival since we already had it). In this simulation we made a flow of people (coming from North-East, where the entrance of the area is) gather around the stage using the automatic mechanism described earlier. As can be observed, the first agents to arrive prefer the areas on the same side of the stage (the larger amounts of agents on the left of the Figure 15), whereas agents that arrive later will cross to the other side due to the amounts of agents already present at the stage (larger flow towards the right side of the Figure 15).

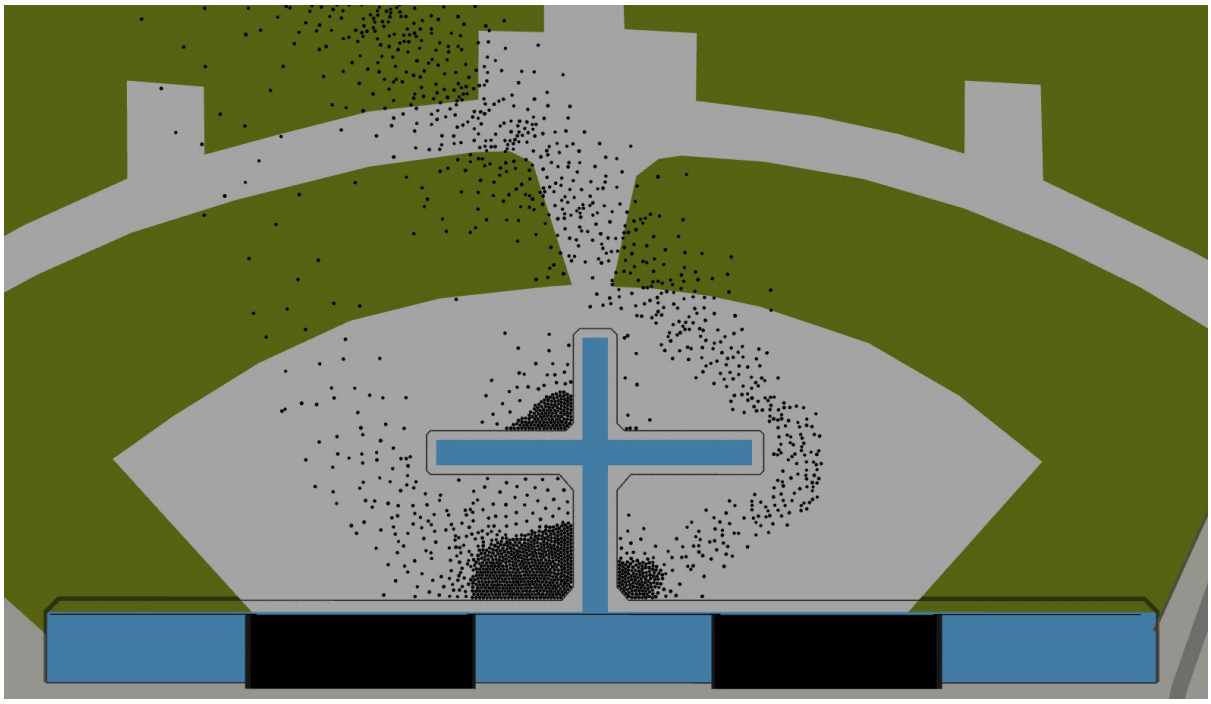

Figure 15. Flow of agents arriving from the North-East and gathering around a cross-shaped stage. The first arrived agents prefer to observe the performance from the same side of the stage (left). Agents arriving later decide to cross to the other side due to the crowd (right).

1.6. Balance Recovery Control

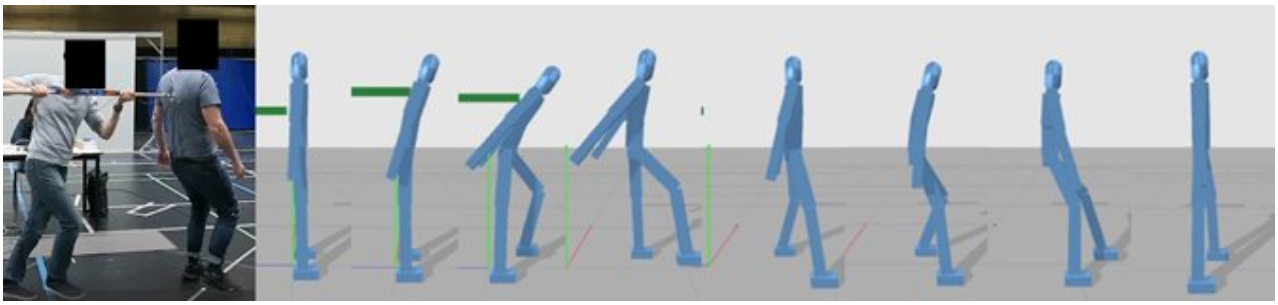

Figure 16. Experiment and simulation of balance recovery after a push from a pole.

1.6.1. Introduction

When standing still, humans rest on their two feet in the upright position. After an external perturbation like a push, they try to recover balance and avoid falling (see Figure 16): several motion strategies are known in humans to achieve this [7]. On a flat ground without any possibility of grip, balance is achieved when the center of mass (*CoM*) lies within the limits of the support area (the convex hull defined by the feet contact points for an individual). However, under large perturbations, adjusting the shape of the support area by repositioning the feet is required. This stepping strategy combines controlling feet position together with the *CoM* position to get the balance condition satisfied.

Our objective is to dynamically control the movement of a character to simulate balance recovery behaviors after external pushes by replicating the stepping strategy of humans. Our approach is to control the character's joints to place the projected *CoM* in the center of the support area, while the feet are re-positioned when necessary to adjust the configuration of this zone according to the instantaneous movement of the *CoM*.

We propose a physics-based simulation that implements this control strategy on a character. We adapt a generic walking model, enhancing its capabilities to handle external pushes and to react accordingly. Our approach is

supported by experiments performed on balance recovery behaviors in humans. This study highlighted the role of the feet in balance recovery and supported our paper in two aspects. Not only it provided us with generalized equations describing the feet behavior, it was also used to fine tune the overall model response with regard to experimental data. In comparison with other learning or kinematics-based approaches that rely on example motions, our approach is generic and can consider new perturbations or new morphologies without need for additional data.

This paper has two main contributions. The first contribution introduces a balance assessment method able to detect the need to take a step to maintain this balance after a push. The second contribution introduces a novel foot positioning strategy leading to a balanced state. These two contributions are combined with state-of-the-art physics-based human simulation, then tuned and validated using the same experimental data that inspired the control of the character.

The result is a simulator able to reproduce the behavior of a human being pushed accurately, for any push strength or duration.

1.6.2. Overview

Figure 17 depicts the method used to simulate balance recovery, with a visual representation of the character being pushed and a flowchart of the simulation iteration. We follow the SI metric system for forces, distances and velocities. The legend of the figure shows the correspondence between the colors of the flowchart boxes and the main components of the proposed method: the Locomotion System which trigger steps to recover balance, the control of the Foot Trajectory after steps are triggered, and finally the FullBody Control of the Center of Mass (CoM) position. Following paragraphs provide details about each of these components, starting with the controlled character itself.

Character. The character of the simulation is a poly-articulated kinematic chain of cuboid limbs, except for a capsule head. A total of 14 limbs are attached at 11 joints. Of all the joints, 4 hold 1 Degree of Freedom (DoF) while the others have 3, amounting to a total of 25 DoFs. The size and mass of the limbs in the body follow Winter's [34] table, scaled with the character's total height and weight.

Joint PD controller. Actuation of the motion for the character is mainly done through traditional Proportional Derivative (PD) controllers. For every joint, a unique set of Kp and Kd gains is provided handling all the DoFs. As seen in Figure 17 close-up 1, desired and current angles θd and θ are compared in a local limb aligned basis. Taking also into account the angle velocity θd and θ, the PD controller outputs a torque τ actuating the joint to the goal angle, according to the classic PD controller equation (1).

$$\tau = K_p.(\theta_d - \theta) + K_d.(\dot{\theta}_d - \dot{\theta}) \quad (1)$$

Simulation iteration. At the start of the simulation iteration, the current state of the character is evaluated by the Locomotion System (Section 1.6.4) in red. This process relies on analyzing the Center of Mass (CoM), the Expected Center of Mass ($XCoM$) and the Base of Support (BoS). If the two feet are planted (i.e., the character is not walking yet), balance is assessed. If necessary, a step is triggered putting the character in the walking state.

Depending on this analysis, a step might be necessary, shown by the Foot Trajectory component displayed in blue in Figure 17. This entails first computing a goal position coordinates for the swinging foot on the ground XZ plane, here known as $Xdesired(XD)$ and $Zdesired(ZD)$, that will lead the character to balance. To reach this goal, a trajectory is computed to be followed during the next simulation iterations. An overall motion is generated for the legs using inverse kinematics, following the trajectory for the swinging foot. Every joint in the body have default goals, some of which are overwritten by the inverse kinematics process when walking.

Those goal angles are provided to the PD controllers, producing a torque τ at every joint of the body. All the forces applied in our method by the Full-body Control (Section 1.6.6) are shown in green. To support the motion of the body, the Jacobian Transpose method is used to compute torques for the three type of forces, $FCoM$, $FVEL$ and $FGRAV$. An example is provided in Figure 17 close-up 2, where the force of limb

i, $FGRAVi$, is applied through two torques $\tau 1$and $\tau 2$ using the elbow and shoulder joints, according to their Jacobian matrix in relation to each limb's Center of Mass.

All those torques are then merged and integrated in the physics engine (using the Dantzig-Wolfe [9] decomposition solver applied to Mixed Linear Complementary Problems), for a new iteration to begin. This process can be disrupted by an external force $FPUSH$, directly applied to the body during the integration.

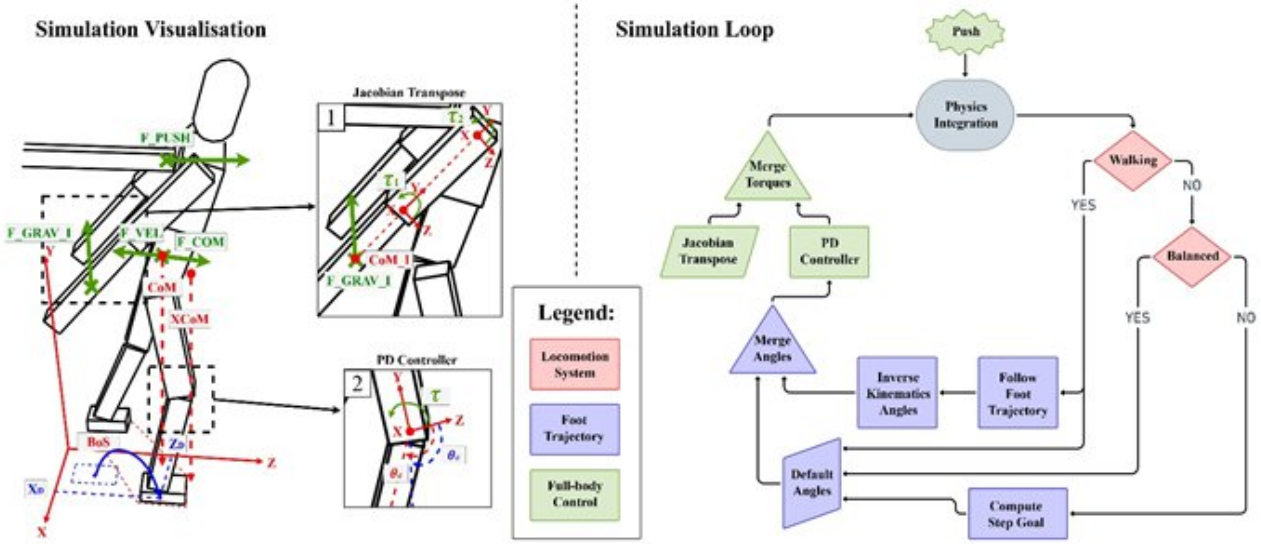

Figure 17. Overview of our balance recovery method.

1.6.3. Feet Motion Strategy

The core of this work relies on the simple premise that humans, after receiving a push, may need to take steps to recover balance. At the core of this recovery is the contact between the character and the ground: their feet. This section details the two parts of the model involving the placement of the feet, respectively the Locomotion System and the Foot Trajectory.

1.6.4. Locomotion System

The first objective of the simulation is to accurately detect whether it is necessary or not for the character to take a step. This section introduces how the current state of the simulation is analyzed, leading if necessary to a walking state. It also explains the state machine followed by the model, to alternate stepping between legs to recover balance.

Balance assessment using Time to Base of Support. The purpose of this work is for the character to recover and keep balance, with or without locomotion. As such, the first notion to be clarified is the definition of balance: what factual observation can be made to determine whether a character is balanced or not. For this purpose, this work relies on experimental data of people being pushed.

More specifically, the step triggering condition determined in the following work, is implemented in our model. This method relies on the analysis of the future position of the center of mass XCoM [14], and is largely inspired by the following notion of Margin of Stability (MoS)[26], computed with the following equations:

$$XCoM = \mathbf{x}_{\mathbf{CoM}} + \frac{\dot{\mathbf{x}}_{\mathbf{CoM}}}{\omega_0} \qquad \text{with } \omega_0 = \sqrt{g/l} \qquad (2)$$

$$MoS = (u_{max} - XCoM).\frac{\dot{\mathbf{x}}_{\mathbf{CoM}}}{||\dot{\mathbf{x}}_{\mathbf{CoM}}||} \qquad (3)$$

Here, $xCoM$ represents the CoM position, g is the gravity constant, l the length of the legs and $umax$the closest intersection with the BoS in the direction of the XCoM from the CoM. Figure 18 illustrates these

notions and the relationships between them. Those concepts lead to the estimation of the Time to BoS (TtBoS) using the Distance to BoS (DtBoS), in the following equations:

$$TtBoS = \frac{DtBoS}{||\dot{x}_{CoM}||} \quad (4)$$

$$DtBoS = (u_{max} - x_{CoM}).\frac{\dot{x}_{CoM}}{||\dot{x}_{CoM}||} \quad (5)$$

The DtBoSrepresents the distance between the CoMand the closest BoS boundary in the CoM velocity direction, while theTtBoS is the time for the CoM to cover that distance. Based on experimental data, a threshold T is defined for the Time to BoS, under which the character will be considered unbalanced in the future, without possibility to recover without taking a step.

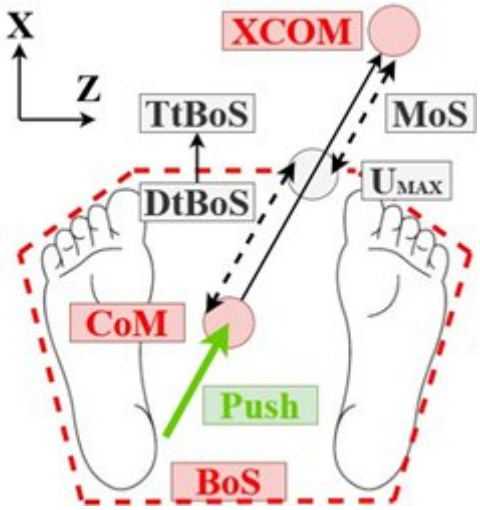

Figure 18. Step detection process when receiving a push (in green)**.**

Leg switch state machine. Using the previously defined threshold T, the simulation is now able to detect whether the character should be stepping or not. This allows us to define a simple state machine which is the locomotion model of the character, illustrated in Figure 19.

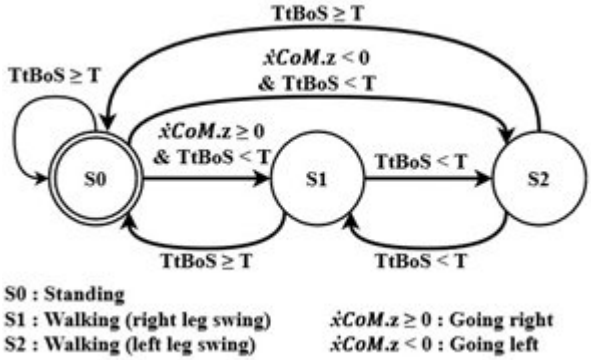

Figure 19. Locomotion state machine.

The base state of the character is the Standing (S0) state. In this state, the rigidity of the character is enough to keep its CoM in the BoS, and steps are not needed. If the threshold T is crossed, a step is initiated. This changes the state of the character as Walking with the right (S1) or left leg (S2) depending on the direction the CoM is moving toward.

While stepping, the characters converge toward a balanced state. However, if at the end of the step when the feet are both at their intended position, the character is still considered as unbalanced (the TtBoS is still under

T), then another step is initiated. The swinging leg is switched, and the process is repeated until the character is considered as balanced at the end of a step. Thus, it simply returns to the Standing state, where internal forces are enough to keep balance.

1.6.5. Foot trajectory

Computation of the foot trajectory. When the character enters a Walking state, first are computed the end goal position's coordinates, XD and ZD, for the foot of the current swinging leg. Not only must the end position be computed, but the StepTime the step takes to be completed is also crucial for swift yet cohesive movement.

In order to calculate those key factors, equations derived from experimental data are being used. Both the foot end position and stepping time have been found to be tightly linked to the velocity of the CoM. Similar to the threshold for the time to BoS, those parts of the model are anchored in observation of experimental data which supports the simulation, leading to the following equations and parameters (see Table 1):

$$(X_d, Z_d) = x_{Foot_{projected}} + a_1.height.\dot{x}_{\mathrm{CoM}_{\mathrm{projected}}} \quad (6)$$

$$StepTime = a_2 + b_2.\dot{x}_{\mathrm{CoM}_{\mathrm{projected}}} \quad (7)$$

Table 1. Experimental Parameters.

Name	$a1$	$a2$	$b2$
Value	0.581	0.185	0.272

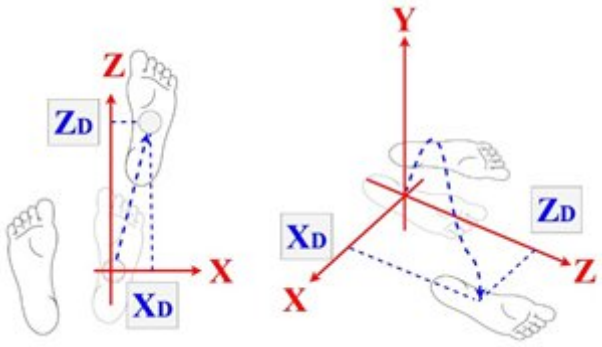

Figure 20. Step Computation in 2D and 3D**.**

In order to reach the goal position (XD,ZD) in StepTime seconds, a trajectory needs to be established that will lead the foot to the next position, as seen in Figure 20. The problem can be divided in two parts : the projected trajectory of the foot on the ground over time, and the foot's height over time. The projected position can simply be computed by linear interpolation between the original and end position, seen on the left of Figure 20. For the height, the model approximates it using a spline proportional to the step distance, resulting in a 3D trajectory like the right part of Figure 20.

Taking into account the planned StepTime, this results in a trajectory that can be followed over time for the stepping foot. At every simulation iteration, the goal position of the stepping foot follows the trajectory according to the current elapsed time since the beginning of the step, leading to the goal (XD,ZD) position in StepTime seconds.

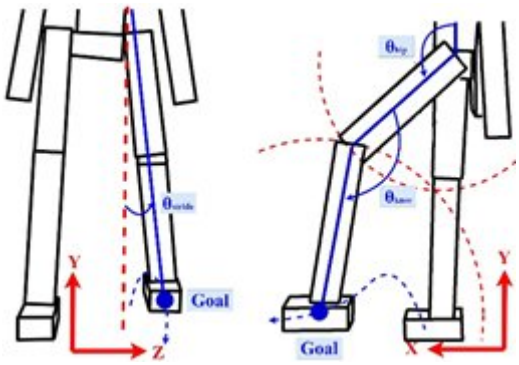

Figure 21. Two-step Inverse Kinematics

Swinging leg inverse kinematics. By default, each joint in the character's body has a target angle of 0 for every DoF while being subjected to gravity. When a step is underway, those goal angles are overwritten for the lower body to be in the right position. Those angles are computed at every simulation iteration when in the Walking state, following the trajectory previously established.

The angles are calculated using the Cycling Coordinate Descent (CCD) algorithm, as seen in Figure 21. The first step is in the coronal plane, to compute the rotation around the X axis $\theta stride$ which relies on the single DoF of the knee to simplify the inverse kinematics to a plane. The goal in blue follows along the trajectory, represented by the dashed line. This is shown in the second part, where by knowing the length of the leg's limbs and the position of the goal, the position of the knee can be deduced and the angles θhip and $\theta knee$ are calculated.

Standing leg inverse kinematics. To support the stepping motion, the standing leg must contribute to the balance. The goal position of the CoM is estimated as being between the final position of the two feet when they are projected onto the ground. The standing leg is given goal angles with the foot not moving and the CoM in the correct position, using the same CCD method but reversed (with the end effector now being the hips).

1.6.6. Full-body Control

Across the whole body, movement is at this point generated by the joints' PD controllers provided with goal angles. Those goal angles are at default 0. As such, the overall body's angles only stem from the weight of gravity on the limbs counteracted by the PD controllers. Down the waist, angles are redefined when stepping. This however leaves the rest of the body in its default state, which is both inefficient and incorrect. When pushed, every part of the body is involved in push recovery albeit at different scales. This section covers the computation of the other applied torques across the whole body.

Set of applied torques. A total of four different types of torques can be applied in the simulation. The first is from the PD controllers, producing torques from goal angles. The other three are gravity compensation, velocity compensation and CoMdriving, computed using jacobian transpose. Gravity compensation compensates the constant effort needed to resist gravity, to lower the strain on the PD controllers for finer control. Velocity compensation's goal is to involve the whole body to counteract the push by converging toward zero CoM velocity. Finally, CoM driving supports the legs' PD controller in their stepping motion by pushing the ground-projected CoM toward the middle of the feet.

Torque computation. The method used in order to compute the torques is the jacobian transpose, a classic control method in robotics [5]. Figure 22 illustrates the application of a force $FGRAVi$ at the position $CoMi$ through two torques $\tau 1$ and $\tau 2$, one for the lower arm and one for the upper arm. To compute one of those torques, the first step is to compute for the corresponding limb the jacobian transpose of its joint in relation to the position $CoMi$ with:

$$J_{limb} = \frac{\delta P_{CoM_i}}{\delta \theta_{limb}} \quad (8)$$

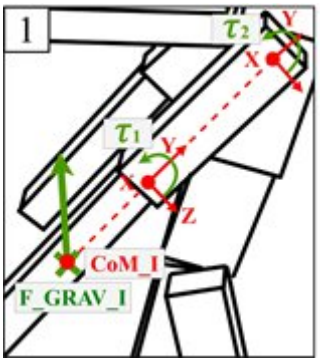

Figure 22. The Jacobian forces applied.

Transposing this limb's jacobian matrix, a torque can be computed for the corresponding limb's joint that will apply the equivalent of a force F ($FGRAVi$) at the position $CoMi$ using:

$$\tau_{limb} = J_{limb}^{T} F_{GRAV_i} \quad (9)$$

The implementation of this process and the relevant forces values to add to the system (namely the following gravity and velocity compensation) are thoroughly explained in the paper 'Generalized Biped Walking Control' [8], which is the main inspiration of this work.

Gravity compensation. The first torques applied are gravity compensation. A set of forces is computed, one for each limb of the upper body. The position of force exertion is the CoM of the limb, with the force following the equation:

$$F_{grav_{limb}} = -g.Mass_{limb} \quad (10)$$

For each force in this set, the Jacobian transpose process is repeated. The joints between the corresponding limb and the hips have a torque computed. As such, the shoulder joint not only compensates the gravity of the upper arm, but also the lower arm as this joint influence both limbs.

Velocity compensation. The next force applied is velocity compensation, at the CoM of the character, with the corresponding equation:

$$F_{VEL} = -a_{VEL}.\dot{x}_{CoM} \quad (11)$$

$aVEL$ is a dimensionless tuning parameter, to regulate the intensity of the resulting torques. This force is only applied once at the CoM, however torques are computed for every joint between the standing foot and the head.

CoM driving. As an addition to the previous set of forces, another force called the CoM driving contributes to balance. It is applied the same way as velocity compensation, from the standing foot to the head, and only once at the CoMof the character. It is calculated using:

$$F_{CoM} = a_{CoM}.(x_{CoM_{desired}} - x_{CoM}) \quad (12)$$

$aCoM$ is another dimensionless tuning parameter, once again regulating the final torques values. It is a mixture of the two previous forces: it supports the PD controllers like the gravity compensation and involves the whole body like the velocity compensation.

Force purpose and equilibrium. The three forces added to the system are expressed the same way as the PD controllers : through torques at joints of the body. This means that at each joint, multiple torques might be summed to get the final exerted torque. In order for this summation to work, each torque value must be carefully tuned in order to ensure a balanced contribution of controls, through gains (Kp, Kd) or tuning parameters ($aVEL$,$aCoM$).

1.6.7. Balance Recovery Results

After the optimisation of the simulation, several aspects of the resulting movement have been explored. First, the balance recovery capacities of the model with regard to the external pushes' intensities were investigated. Second, the torques generated by the model were investigated in terms of intensity and compared to human strength generation capabilities reported in the literature. Third, the generated motion was compared to reference values obtained from the experimental data and processed with the CusToM motion analysis library [23]. Specifically, step triggering, CoM trajectories and joint angles trajectories were compared in terms of amplitude and shape. The simulation runs in real time between 100Hz and 200Hz. Each frame represents 0.005 seconds to fit the experimental data refresh rate, with 200 solver iterations per frame.

1.6.7.1. Evaluation of balance recovery

The first goal of this simulation is to mimic the balance recovery behavior, and more specifically how humans follow stepping strategies when necessary. As such, the first qualitative metric explored in Figure 23 is the ability of the character to recover balance after receiving pushes of varying strength and duration. By default, we set our character to an average height and weight (1.70m and 70kg), pushed from a perpendicular angle to the back. A push is done at every 20N (between 0 and 300 Newtons) and 0.2s (between 0 and 2 seconds), only once due to the deterministic nature of the simulation. Note that this range of impulse goes beyond the experimental study we based our work on, as shown in Figure 23. The blue and red areas correspond to complete success or failures. Mixed responses designate the area in which failed test appeared for lower intensity pushes than successes.

1.6.7.2. Torque value

The following investigation aims at checking whether the generated torques values are within the range of human torques. The measures are made for the lower body joints, and compared with an empirical study [3]. Figure 24 illustrates our results. For each type of torque, the maximum torque (up axis) observed in humans for an angle (right axis) and an angle velocity (bottom axis) value is represented as a surface. Each color corresponds to the torque measured over the duration of a low (green), medium (blue) and high (red) impulse push. The surface represents the torque generation capacities for the corresponding angle and angle velocity values reported from Anderson et al [3].

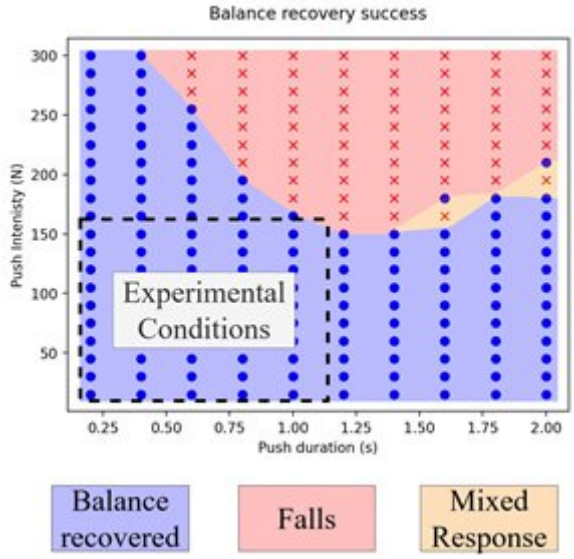

Figure 23. Balance recovery for varied impulse, with the range of intensity and duration covered experimentally delimited.

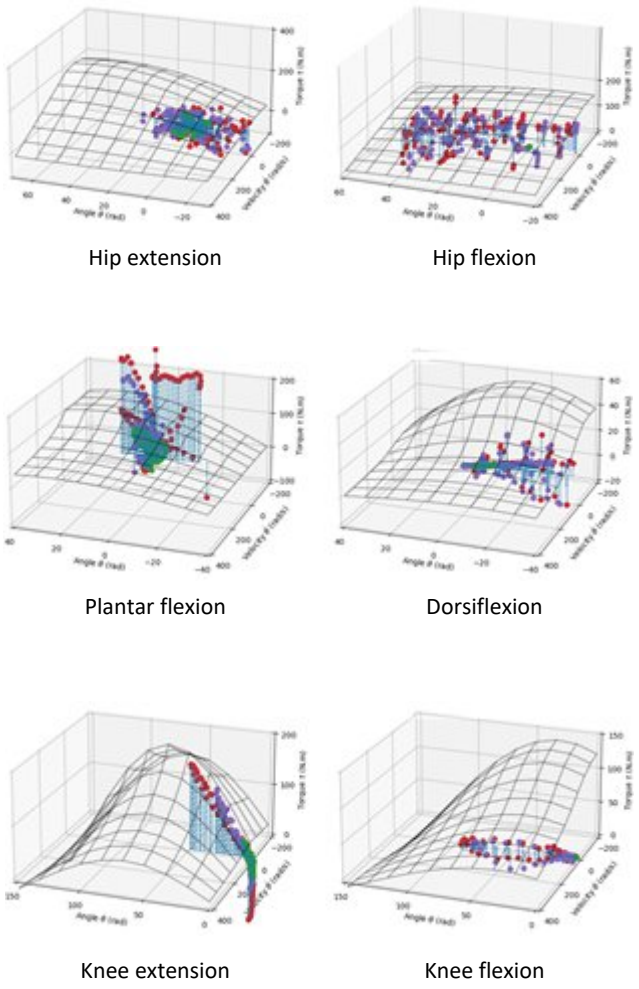

Figure 24. Simulation torque comparison**.**

1.6.7.3. Data comparison

In this subsection, the simulation results are confronted and compared to the experimental data computed from measured human pushes.

Step triggering

The confusion matrix Table 2 of the simulated experiment pushes allows an evaluation of the step triggering condition used in the model. For every push perpendicular to the back of the dataset, the same push was recreated in the simulator. The test is done on whether the simulation needs to take a step or not for the same pushes than the original data. Positive and Negatives correspond to the simulation needing a step, while True and False come from the experimental data needing to step. The resulting values are a sensitivity of 0.88, a precision of 0.81, and an overall accuracy of 0.74.

Table 2. Step triggering confusion matrix of 251 pushes.

	True	False
Positive	163	38
Negative	22	23

CoM trajectory

Figure 25 (left) showcases a study of the CoM trajectory over time. Every push at a perpendicular angle of the back in the original dataset (blue) is recreated in the simulation (red). Only stepping pushes are shown in this figure. The trajectory is the projected position of the CoMon the ground over time (normalized by height and weight for the data). Every push of the experiment is recreated in the simulation, with the color gradient corresponding to the intensity of the same push for both set of trajectories.

The final positions in the direction of the push after stepping (by impulse) are displayed in Figure 25 (right), for the experimental data (blue) and simulation (red). All pushes perpendicular to the back are tested, regardless of steps in the experimental data.

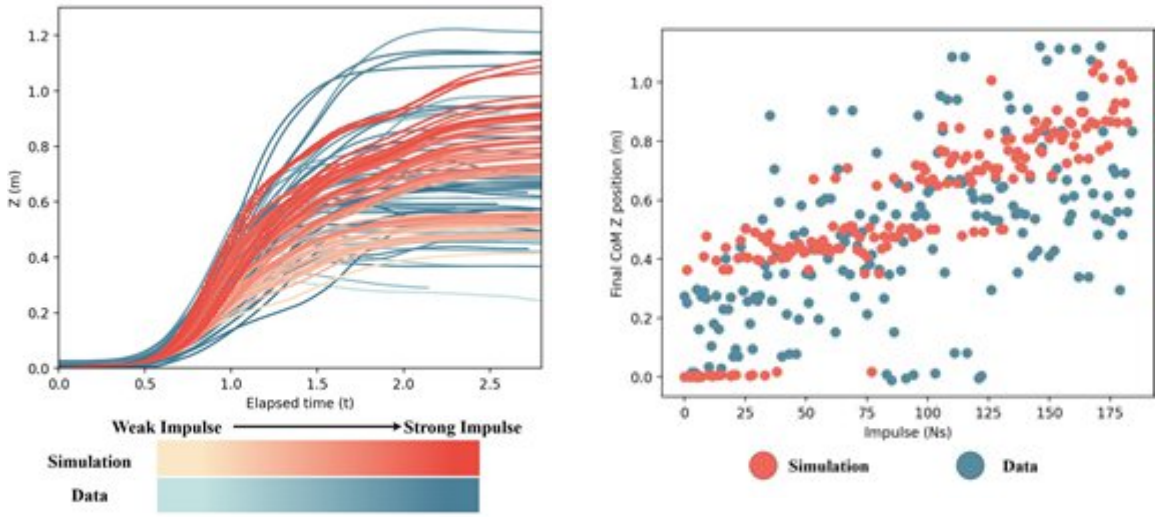

Figure 25. Final positions in the direction of the push after stepping.

Conclusion

In this deliverable, we have described the different components of the first version of the CrowdDNA crowd simulator, which was developed during Period 1 and Period 2 of the project. Our proposed solution is not limited to just modeling the macroscopic behavior of a crowd, but it also models many micro-scale effects such as 3D volumetric characters, close character interaction, balance control, as well as improvements at the macroscopic behavior based on social and contextual cues.

This new simulator provides many key functionalities that will be essential for the upcoming parts of the project. With the ability to model both macro and micro, we will be able to train models for macro-to-micro crowd analysis that will enable an unprecedented control and management of dense crowds in real-time. Leveraging the proposed model, we will generate a new richly annotated dataset of 3D crowds, with micro-level annotations such as inter-human contact forces, limb-level actions, and individual behavior. This new type of data will enable the training of machine learning models for crowd analysis and management, planned for the period 3 of the project. In the final part of the project, the trained models will be deployed and tested to a range of crowd observatories to validate the capabilities of solutions proposed along the project.

References

Van Toll, Wouter and Grzeskowiak, Fabien and Gandía, Axel López and Amirian, Javad and Berton, Florian and Bruneau, Julien and Daniel, Beatriz Cabrero and Jovane, Alberto and Pettré, Julien. 2020. "Generalized Microscropic Crowd Simulation using Costs in Velocity Space." *ACM/SIGGRAPH Symposium on Interactive 3D Graphics and Games.*

Jur P. van den Berg, Ming C. Lin, and Dinesh Manocha. 2008. "Reciprocal Velocity Obstacles for real-time multi-agent navigation." *Proc. IEEE Int. Conf. Robotics and Automation.*

Stephen J. Guy, Jatin Chhugani, Sean Curtis, Pradeep Dubey, Ming Lin, and Dinesh Manocha. 2010. "PLEdestrians: A least-effort approach to crowd simulation." *Proc. ACM SIGGRAPH/Eurographics Symp. Computer Animation.*

Karamouzas, Ioannis, and Mark H Overmars. 2010. "A velocity-based approach for simulating human collision avoidance." *Proc. Int. Conf. Intelligent Virtual Agents.*

Loper, Matthew and Mahmood, Naureen and Romero, Javier and Pons-Moll, Gerard and Black, Michael J. 2015. "SMPL: A Skinned Multi-Person Linear Model." *ACM Trans. Graphics.*