2025 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW) | 979-8-3315-1484-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/VRW66409.2025.00148

Simulating Body Movements for Multiple Agent Avoidance

Ulan Akmatbekov, Janis Sprenger*, Rui Xu, Philipp Slusallek
German Research Center for Artificial Intelligence (DFKI)
Saarland Informatics Campus

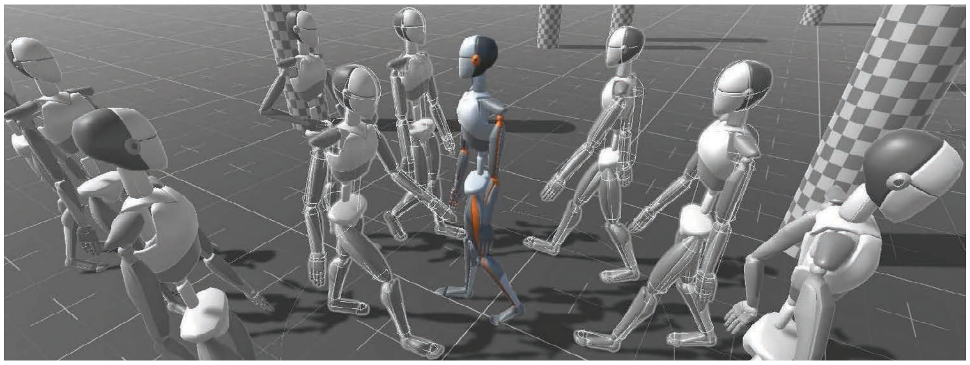


Figure 1: Our system can animate and control over 20 interactive agents on standard gaming hardware.

Abstract

Animation of virtual characters in a crowd is one of the unavoidable hurdles artists and developers encounter throughout their work in different fields, such as cinematography, simulations, and game development. Depending on the requirements for the quality of motions, various techniques are used to overcome this constantly rising problem, each with its own shortcomings. Ignoring the interaction between individuals by using simple hitboxes to avoid collisions produces unrealistic movement patterns, while the manual animation of characters is a tedious and costly endeavor. This work discusses the implementation of an automated system capable of realistic human motion synthesis for multiple avatars, focusing on the interaction between individuals. The method is based on an autoregressive, data-driven conditional variational autoencoder and an additional control policy providing the latent input vector for the motion network trained with reinforcement learning. Different virtual sensor types that perceive a variable number of individuals and obstacles around the agent are proposed and evaluated. The system provides the simulation of the body movements in a multi-agent environment. It can operate at runtime with an adequate frame rate in environments containing more than 20 characters.

Index Terms: Data-Driven Animation, Motion Capture, Reinforcement Learning, Crowd Simulation.

1 Introduction

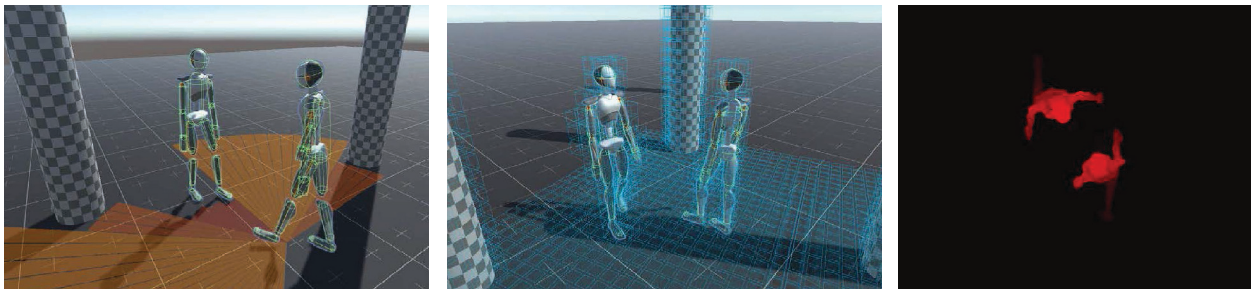
Navigating virtual characters through a crowd is a long-lasting and well-established research area (we refer to [17, 31] for reviews). When characters pass each other, their meshes should neither collide nor should they stay unreasonably far away, giving the impression of a virtual wall separating them. Therefore, the animation must visualize the evasive movements in addition to the navigation [13], and thus, classical approximations of the characters using an elliptical shape [4] are not sufficient. Particularly in virtual-reality (VR) applications, any system must be responsive and lightweight. Existing approaches do not coordinate the movements [1] require extensive computation time [7] or do not scale to arbitrary crowd sizes [27, 29].

In this work, we extend a motion VAE [21] to generate avoidance movements while following a navigation control input. The model requires only the motion capture data of a single person showing avoidance movements and can generalize to different numbers of "opponents". While the animation is generated using a data-driven approach, the control is exerted with a separate control network trained with reinforcement learning. Unlike [18], we do not focus on the simulation of crowd behavior but on coordinating body movements and trajectory adjustments to avoid other entities inside a crowd. The performance and practicability of different input types to perceive other characters are evaluated. Our target model can generate plausible evasion movements when avoiding other agents in the scene with a low computational cost and no adjustments to the navigation system required. Thus, it can be utilized to animate (non-player characters) NPCs in VR games.

2 Character Animation in Crowds

Manually creating appealing and convincing avoidance animations with finite state machines or blend-graphs requires significant effort and creative skill, which is not always feasible for generic NPCs [11]. Established methods like motion matching [6, 12] reduce the effort required to create the animation graph, but fine-tuning the system to show natural evasion movements remains difficult. Data-driven approaches utilize motion capture data to create a generative motion model synthesizing new animations in a given environment. Several recent approaches have considered interactions between characters (e.g., [20, 28]) with incredible performance and plausibility, especially for locomotion tasks. However, most interaction simulations are primarily focused on two single individuals (e.g., [9, 22, 27]) and require paired data-sequences for the proper simulation of interaction (e.g., [8]). Such an approach is not practical for crowd simulation, where multiple people are interacting with each other. Approaches considering more than one opponent are rare and usually restricted to a fixed number of opponents (e.g., [29]) and still require multi-person capture data. While diffusion-based approaches [26] show promising results for trajectory following and can be used in an optimization scheme [7], their

*e-mail: Janis.Sprenger@dfki.de



(a) Sector perception (b) Voxel perception (c) Depth camera perception

Figure 2: A visualization of the different observation sensors utilized in this work.

computational effort exceeds the available resources in VR games.

Physics-based motion models utilize optimization (e.g., [19]) or reinforcement learning (RL)-based control policies (e.g., [3, 24]) to control a physical representation of the human body. As a result, the model can react realistically to external perturbations, like bumping into someone else, as recently demonstrated for dense crowds [10]. Extending the lower-body animation with upper-body evasive actions is feasible [1], but the coordination between lower and upper-body movements is missing. Novel simulation models like ASE [25] and motion VAEs [21] combine a general-purpose skill network - trained to generate animation poses - with a high-level action policy providing the control input to the skill network via a latent code. Thus, the control policies, which are trained with reinforcement learning, exert full body control indirectly via the skill networks. However, multi-person interaction has not yet been integrated into these models. In a VR application, the physical interaction of NPCs in crowds with the player is not always preferred and does not provide additional benefits without haptic feedback. It is thus questionable whether the computational effort of a physics-based approach is justified.

3 Architecture

The model architecture consists of two neural networks: a motion model (skill network) and a control or action policy. The motion network is constructed as an autoregressive conditional variational autoencoder, predicting the next pose based on the past pose and a latent control input. The control policy generates a control input based on the environment observation and navigation information.

3.1 Motion Generation

Pose Encoding. A single pose is encoded in root local space, with the origin at the hip location projected to the ground plane rotated to the agents facing direction. A single pose consists of the root position and rotation displacement ($\dot{r}^x, \dot{r}^z, \dot{r}^a \in \mathbb{R}$), the joint positions ($j^p \in \mathbb{R}^3$) and velocities ($j^v \in \mathbb{R}^3$) as well as the forward and upward vectors of the joint rotation ($j^r \in \mathbb{R}^6$) in root-local space.

Motion Model. The motion model is trained as a conditional variational autoencoder (cVAE). The encoder network $\Phi(z_t|x_{t+1}, x_t)$ compresses the pose of the next frame x_{t+1} with respect to the current pose x_t to a latent motion manifold z_t. The decoder network $\Omega(x_{t+1}|z_t, x_t)$ is formulated as the conditional probability distribution of a character pose at the next frame x_{t+1} given the current pose x_t and the latent control variable z_t.

Control Policy. A control policy $\pi(z_t|s_t)$ is trained using principle policy optimization (PPO) in a reinforcement learning (RL) setup to control the motion generation via the latent control vector z_t based on an observation s_t. The policy parameters are optimized to increase the reward for actions generated on a given state of the environment.

3.2 Virtual Observation Sensors

The observation contains the control input (directional velocity, closest path point, contact flag) in addition to the perception of other agents. Three different perception models were evaluated in this work. Figure 2 shows visualizations of the perception modules.

Sector Perception divides the area in front of the agent into discrete sectors on a circular plane. Each sector represents the closest distance to a dynamic or static obstacle projected to the ground plane similar to [5]. The field of view (FOV) is rotated with the agent's movement and covers 220° as an approximation to the human FOV, and the distance is limited to three meters.

Voxel Perception provides a volumetric perception in a cuboid volume around the agent similar to [28], which denotes the presence of a physical object within its bounds at a certain frame t. The voxel space for a given number of voxels on each axis (N_x, N_y, N_z) can be defined as $V_t \in \{0,1\}^{N_x \times N_y \times N_z}$

Depth Camera Perception provides environment vision with a top-down depth camera observing the surroundings of the agent. Unlike a head-mounted camera, a top-down camera provides additional information in areas not directly perceivable by the agent.

Other perceptions have been evaluated but were considered insufficient for the target application. A single front-facing camera similar to [1] and a finite number of rays cast from the head perspective did not provide enough contextual input for a stable simulation and contained many blind spots. Multi-camera and panoramic cameras were computationally expensive and thus not applicable for NPC simulation.

4 Data Acquisition

We captured our own data using an Xsens Awinda full-body capturing system with 17 wireless inertial measurement units (IMUs) in a 3.5 x 7m capturing area and two motion actors from the research group. In addition to random locomotion with stopping, idling, and walking ($1.4 - 1.9\ m/s$), two traffic cones were positioned to enforce an infinity symbol trajectory to encourage more encounters. The actors were asked to change their gate (stop, idle, walk) and not bump into but avoid each other as close as possible. A visualization of this scenario can be found in Figure 3. Although two capture suits were utilized to investigate and visualize the movements, only the data of one actor was used for the training.

The raw motion capture software was first processed with the MVN Awinda software of Xsens to reduce sensor drift, foot skating, and magnetization. The resulting BVH-Animation data was further cleaned of any artifacts and additionally mirrored along the sagittal body plane to reduce motion bias and increase the amount of training data. The motion data was encoded and exported as training data sequences. A total of 38,116 frames at 60 frames per second or 10.6 minutes of motion capture data was utilized for training.

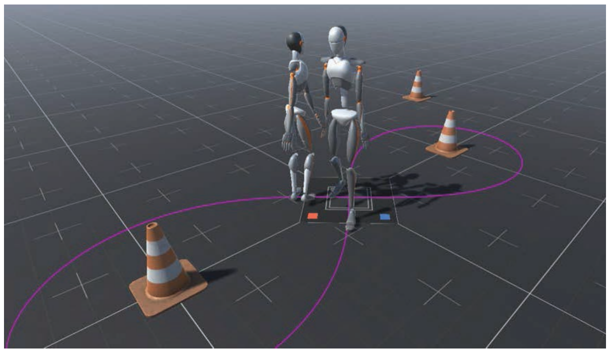


Figure 3: Example data of the avoidance capturing. Virtual cones and trajectories visualize the actual positioning of cones and target trajectories in the real capturing environment.

5 Implementation

5.1 Motion Model

The motion model is implemented in Pytorch [23] using the implementation of [21] as a basis. The motion encoder Φ is implemented as a three-layer fully connected network with a hidden layer size of 512 and exponential linear units (ELU) as an activation function. Reparameterization [16] is applied to circumvent the non-differentiability of the stochastic model, and the resulting latent action control vector z_t has a length of 32.

The motion decoder Ω is a mixture of expert model [32] using six expert networks and a single gating network. The input to the decoder is the concatenation of the latent z_t and the previous pose x_t. The gating network is a three-layer fully connected network with ELU activations. The hidden layer size is 64, and a softmax operator ensures that the sum of the expert weights equals one. For each iteration, the expert weights are used to create a specialized network $\bar{\Omega}$ by combining the weights and biases of the expert networks in a weighted sum. The resulting weights are utilized to execute the three-layer decoder using the ELU operator and a latent size of 512

The network weights are optimized with stochastic gradient descent using the Adam algorithm [15] in minibatches of 64 data points. Learning rate decay is applied starting at 10^{-4} and decaying to 10^{-7}. The training data is normalized using z-score and group normalization is applied [30] with the groups being the character displacement, joint positions, velocities, and rotations. Scheduled sampling [2] is utilized to gradually prepare the motion model for autoregressive motion prediction in three stages. The model is first trained for 20 epochs with ground truth data only (teacher stage). In the ramping stage, the probability of using the autoregressive prediction is linearly increased from zero to one over 20 epochs. Last, the model is trained for 100 epochs in a pure autoregressive stage. In each stage, the network performs 8 regression steps. The reconstruction error is penalized with the mean squared error (MSE), and the variational autoencoder is constrained via β-VAE training using Kullback-Leibler (KL) loss and a weight of $\beta = 0.01$.

The motion model was trained on an Nvidia A100 GPU with 80 GB VRAM, and training was concluded after approximately six hours. The motion model was exported in the open neural network exchange (ONNX) format and integrated within a Unity 3D environment using the Microsoft ONNX-Runtime library.

5.2 Control Policy

For the training of the action policy, the Unity ML-Agents Toolkit [14] is utilized. The sensors and motion generation pipeline are implemented within a custom Unity environment. Thus, the policy training does not change the motion model's network weights. The default navigation mesh implementation inside of Unity, which only considers static but not dynamic objects (like the opponents), is utilized to provide the navigation input, consisting of the directional velocity $\vec{v}_t = (v_t^x, v_t^z)$ and the direction $D_t = (d_t^x, d_t^z)$ to the closest point on the ideal path truncated to uniform length in the root local space. Additional input is provided by the equipped perception system $(S_t|C_t|V_t)$ and an observation flag $O_t^{S|C|V}$, indicating whether an obstacle is perceived.

Several rewards and penalties are applied to control the training of the action policy and form the cumulative reward R_t for every frame. The individual components of this reward are described in more detail subsequently. All sub-rewards are bound and scaled to proportion. Hence, the cumulative reward per frame is primarily in the $[-1,1]$ range. The total reward is thus the sum of all rewards:

$$R_t = R_t^{ppr} + R_t^{dir} + R_t^{tar} - R_t^{div} - R_t^{prox} - R_t^{st}$$

5.2.1 Path Following Rewards

The following rewards are applied to incentivize the agent to follow a predefined trajectory on individual subgoals.

The path point reach reward R_t^{ppr} is a one-time reward of 10 when the agent gets closer than 0.5 m to the next path destination point.

The direction reward is applied to encourage walking directly to the target by reinforcing the factual velocity $\hat{v}_t$ and factual movement angle $\hat{v}_t^a$ to be close to the desired velocity v_t and movement angle a_t, bound by $[f_l^{dir}, f_u^{dir}]$ with the empirically derived values of $[-0.15, 0.25]$.

$$R_t^{dir} = \left(f_u^{dir} - f_l^{dir}\right)\left(\left(\frac{\cos\left(\hat{v}_t^a - v_t^a\right)}{2}\right)^4 \cdot e^{-\|\|\hat{v}_t\| - \|v_t\|\|}\right) + f_l^{dir}$$

The target approach reward is a continuous reward to encourage moving towards the next goal by computing the difference $l_t^g = \|g_{t-1} - r_{t-1}\| - \|g_t - r_t\|$ between the previous and current distances to the destination point g_t. To bound and scale the reward to the range of $[-0.25, 0.25]$, a hyperbolically shaped function A with the sensitivity factor $\alpha^{tar} = 400$ is applied:

$$A(x,\alpha) = 1 - \frac{1}{1 + \alpha * x}$$

$$R_t^{tar} = 0.25 \cdot \frac{|l_t^g|}{l_t^g} \cdot A\left(|l_t^g|, \alpha^{tar}\right)$$

Path divergence is penalized if the agent diverts from the dedicated path. It is again bound by function A and scaled to a range of $[0, 0.2]$:

$$R_t^{div} = 0.2 \cdot A\left(\|d_t\|, 1\right)$$

5.2.2 Agent Avoiding Rewards

During avoidance maneuvers, the agent should not collide with the opponent but keep as close to the original trajectory as possible. Simple one-time collision penalties and full-body pillbox colliders were insufficient and did not create the desired results. Hence, multiple colliders were created for individual body parts (e.g., legs, arms, torso, etc.).

The proximity penalty is computed using the minimal distance between two body parts of opposing agent s_{ij}. This distance is computed for every collider in the agent body in the set $i \in C^{body}$ to every scene object within the radius r around the agent in the set $j \in S^{obj}$ containing the body colliders of other agents. While this is computationally more expensive, it is only required during training

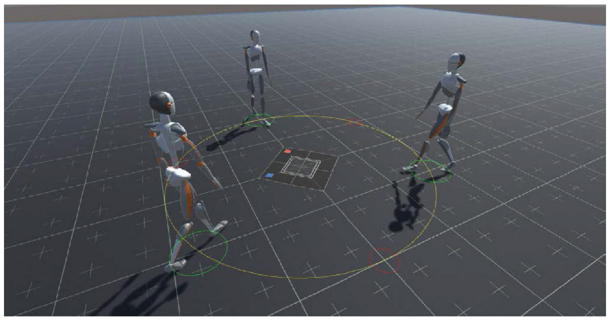


Figure 4: Concurrent avoidance environment with three agents. The agents are spawned on the green start circles and are trained to reach the red target circles on the opposite side, while avoiding collisions in the center.

and not during inference. Quadradic decay is applied to the loss to avoid over-penalization of far away bodies:

$$R_t^{prox} = 1.5 \cdot \frac{1}{1 + 10 \cdot \left(\min s_{ij}\right)^2}$$

The shoulder turn penalty encourages upper body movement for avoidance by projecting the agent's shoulder positions on a line $\vec{l}_t^s$ orthogonal to the movement direction on the ground plane. The distance is first scaled proportionately to the maximum shoulder width $L_{\max}$ and bound to the range $[0, 0.5]$.

$$R_t^{st} = 0.5 \cdot \min\left(\max\left(1 - \frac{|\vec{l}_t^s|}{L^{\max}}, 0\right), 1\right)$$

5.3 Training Environments

Three different training environments are utilized. The first **walking** environment contains a single agent. A random destination point is selected, and upon reaching the point, the agent does not receive any command for three seconds to encourage idling. After the idle period or the maximum number of steps is exceeded, the agent is respawned at a different location, and a new target point is sampled. In the **concurrent avoidance** environment, multiple agents are spawned on a circle with the target point on the opposite side, forcing the agents to pass the circle's center and thus create collision events. The number of agents is varied between $[2, 4]$, the radius of the circle between $[2\text{ m}, 5\text{ m}]$, and the heading angle between $[0°, 180°]$ with respect to the center of the circle. Figure 4 shows an example of this environment. Although many collisions occur in this scenario, it is fairly predictable. A **random points avoidance** similar to the walking environment but with five agents in parallel is utilized to reduce predictability while not containing as many collision events as the concurrent avoidance environments. In our experiments, the most optimal combination of environments was four walking environments, four concurrent avoidance environments (two, two, three, and four agents), and one random point avoidance environment with five agents, resulting in the parallel training of 20 agents.

The training of a single policy differed between the environment input model. While most policies required between 10:40 and 14:50 hours for training 20e6 iterations, the camera perception model required 27:32 hours for 10e6 iterations and thus was not trained for the same duration as the others.

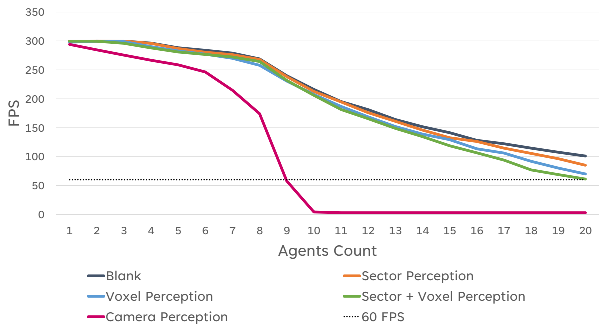


Figure 5: Runtime performance for different numbers of agents and perception models running in parallel on a mid-range gaming laptop.

6 Evaluation

6.1 Runtime Performance

The average frames per second (fps) during inference were recorded in the deployed application with no fps cap and disabled vertical synchronization. The evaluations were executed on a mid-range gaming laptop (Intel i7-6700HQ 2.6GHz, NVIDIA GTX 1070 Laptop, 16 GB RAM). Even with a very small input size of 32x32 px, the depth camera perception agent does not scale. With more than nine agents in one environment, the performance drops below the required 60 fps for interactive applications. All other perception models do scale, and especially with the sector perception agent, it is possible to execute more than 20 agents in parallel at over 60 fps. Even on a high-end gaming PC (AMD Ryzen 9 16 Core CPU, NVIDIA RTX 3080 GPU, 64 GB RAM), the camera agents' performance drops similarly, while 25 sector perception agents run at 135 fps. As this makes the camera-based agent unusable for the target application, it has not been further evaluated.

6.2 Visual Evaluation

All agents show high-quality locomotion animations and can generally steer to a goal position. As expected, a blank agent without the ability to perceive opponents collides with other agents frequently, sometimes showing random evasion movements, resulting in an initially missed target and circling around the target. The sector perception agent displays avoidance movements and upper body rotations to pass by an opponent. It shows that learning to avoid others through spatial body movements is feasible rather than relying on pathfinding skills only. However, there are still instances when a collision occurs, and the system cannot adequately react to suddenly blocked passages. The voxel perception agent slows down and tries to avoid opponents upon perceiving them, but not always successfully. In the case of four opponents, the agent tends to slow down significantly, most likely due to more voxels being triggered in front of the agent, signaling it to stop and wait for a clear path. The combination of sector and voxel perception is unstable and not very agile. While the assumption was that the combination of both perceptions would benefit the agent's behavior, it only resulted in the combination of their shortcomings. In some cases, the body is paralyzed and stops moving, especially during encounters with an opponent. In addition, this model showed the worst behavior regarding reaching the goal position.

6.3 Walking Performance

An objective evaluation of the walking performance was performed by positioning the goal between two to five meters of the agent and rotating the agent 0, 90, or 180 degrees to the goal direction.

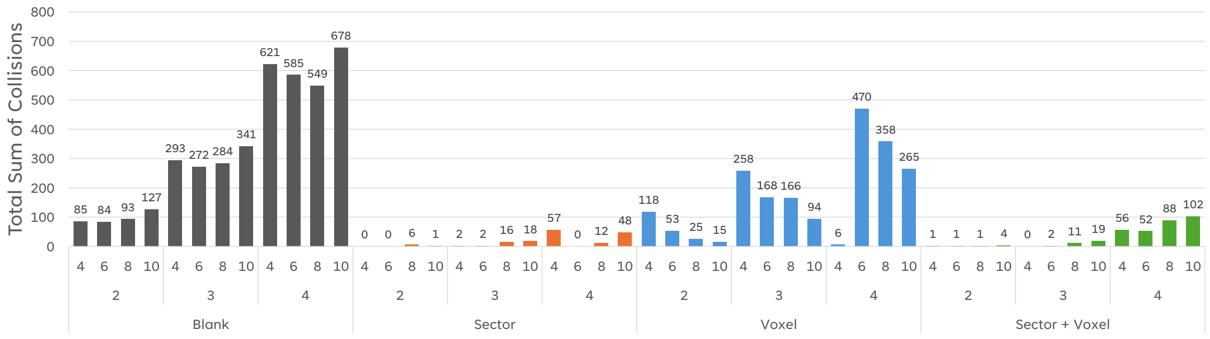


Figure 6: Total sum of collisions per agent for each perception system, number of agents in the scene, and the travel distance.

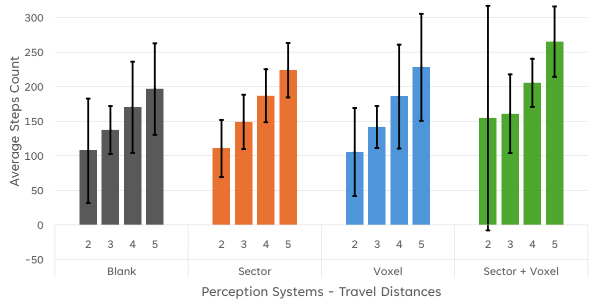


Figure 7: The average number of steps required to reach the goal. Error bars denote one standard deviation. Results are grouped for the agent perception system and the initial distance to the goal.

In total, 1200 trials were executed for each model, and the conditions were balanced over the trials. While the blank and sector agents reached the goal in all trials within the given time window, the voxel agent missed three targets, and the combined sector and voxel agent missed seven. However, not all models are equally efficient in reaching the target, and sometimes, they miss the goal on the first try. Figure 7 shows each agent's average number of frames to reach the target based on the starting distance. While the performance is comparable on average for the sector and blank agents, there is some variance, especially on short distances, primarily due to initially missed targets and subsequent retries to reaching the target by 'circling' around the target position.

6.4 Avoidance Performance

The concurrent avoidance environment with a circle diameter of four to ten meters and two to four agents is utilized to evaluate the avoidance performance. With 100 trials per configuration and each agent within the scene recording data, a total of 3600 trials were executed. While the sector perception agent is able to reach the target in all but one of the trials, the blank agent misses reaching the target within the time limit in 22 trials, the combined sector and voxel agent in 13 trials, and the voxel perception agent in 74 trials.

Even without perception, two agents do not necessarily collide. Small deviations in trajectory, acceleration, and velocity can result in two agents missing each other. On the other hand, agents might collide more than once, especially if the target is initially missed and the agent circles to try it a second time. In the case of a model reaching the target perfectly every time and all agents bumping into each other exactly once, a total of 4000 collisions would be recorded. Collisions are recorded with body colliders, matching individual body parts, not by a simple pillbox collider around the character. The blank agent without perception recorded 4012 collisions during the experiment, including some iterations without collisions and some with multiple collisions. The sector perception agent performed best, with a total of 162 collisions. The voxel perception agent performed poorly with 1996 collisions, and the combination of both sensors did not benefit either (337 collisions). Figure 6 shows the detailed distribution of all collisions for each agent type, the number of agents, and the starting distance. There are no consistent differences between the starting distances, but more agents in the scene generally result in more collisions. Further in-depth analysis shows that the minimal interpersonal distance in the sector perception agents is 0.427 m, being considerably lower than a typical pillbox collider.

6.5 Hyperparameter Evaluation

During the development, several hyperparameters were fine-tuned and tested. While 10e6 training iterations yield sufficient performance in walking, it is insufficient for avoidance movements, and 20e6 iterations improve results significantly. Avoidance skills are lacking with two layers, and four layers require significantly more training without a better performance. During reinforcement learning, 256 steps provide faster convergence to the desired behavior than larger step counts (e.g., 1024) without reducing performance. If the path divergence penalty R_t^{div} is weighted too high (> 1), the model sticks to the path too closely, resulting in more collisions. If the proximity penalty R_t^{prox} is weighted too high (> 2), agents choose very convoluted trajectories and divert significantly from the planned trajectory to avoid collisions.

7 Conclusion

We present a new approach for generating character avoidance movements in multi-agent VR environments using an autoregressive conditional variational autoencoder controlled by an action policy trained with reinforcement learning. The approach requires only a few minutes of motion capture data of a single actor to effectively avoid other characters with minimal implementation effort and almost no adjustments for the navigation planner. Different perception modes to perceive a variable number of opponents have been evaluated with respect to their efficiency and effectiveness. Using a novel perception mechanism, the target model can efficiently simulate over 20 agents, showing avoidance characteristics in our experiments.

7.1 Limitations

The action policy is not yet perfect. It does not always reach the goal point accurately, resulting in undesired behavior (e.g., circling around the goal position). While it displays only a few collisions in our experiments, the model struggles to keep this performance up in highly crowded environments.

7.2 Future Work

While the proposed approach was able to synthesize realistic avoidance movements like upper body rotations, it did not always utilize them. Adding more annotations to the data (e.g., frame-wise annotation with an 'avoid' flag) and capturing more diverse training data could establish a more explicit control over the motion, and different avoidance styles could be introduced. Combining the presented approach with a traditional motion planner as a hybrid system would be highly encouraged in a target application to make the system more robust and always ensure avoidance. By fine-tuning the parameters of the reward function, different types of agents (more aggressive, more submissive, right-handed, left-handed avoidance, etc.) could be trained using the same or a different motion source to increase variability inside a crowd and match the parameters of the agent behavior.

Acknowledgments

This work was funded by the BMBF (project MOMENTUM, 01IW22001), the BMWK (project KAI, 19A21022C), and the Saarland and Intel collaboration (FGM, SL_Intel_03).

References

[1] E. Alvarado, D. Rohmer, and M.-P. Cani. Generating upper-body motion for real-time characters making their way through dynamic environments. *Computer Graphics Forum*, 41(8):169–181, 2022. doi: 10.1111/cgf.14633 1, 2

[2] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, p. 1171–1179. MIT Press, Cambridge, MA, USA, 2015. 3

[3] K. Bergamin, S. Clavet, D. Holden, and J. R. Forbes. DReCon: data-driven responsive control of physics-based characters. *ACM Transactions on Graphics*, 38(6):1–11, 11 2019. doi: 10.1145/3355089.3356536 2

[4] A. Best, S. Narang, and D. Manocha. Real-time reciprocal collision avoidance with elliptical agents. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 298–305, 2016. doi: 10.1109/ICRA.2016.7487148 1

[5] P. Charalambous, J. Pettre, V. Vassiliades, Y. Chrysanthou, and N. Pelechano. Greil-crowds: Crowd simulation with deep reinforcement learning and examples. *ACM Trans. Graph.*, 42(4), July 2023. doi: 10.1145/3592459 2

[6] S. Clavet. Motion matching and the road to next-gen animation, 2016. 1

[7] S. Cohan, G. Tevet, D. Reda, X. B. Peng, and M. van de Panne. Flexible motion in-betweening with diffusion models. In *ACM SIGGRAPH 2024 Conference Papers*, SIGGRAPH '24. Association for Computing Machinery, New York, NY, USA, 2024. doi: 10.1145/3641519.3657414 1

[8] M. Fieraru, M. Zanfir, E. Oneata, A.-I. Popa, V. Olaru, and C. Sminchisescu. Three-dimensional reconstruction of human interactions. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7212–7221, 2020. doi: 10.1109/CVPR42600.2020.00724 1

[9] A. Ghosh, R. Dabral, V. Golyanik, C. Theobalt, and P. Slusallek. Imos: Intent-driven full-body motion synthesis for human-object interactions. *Computer Graphics Forum*, 42(2):1–12, 2023. doi: 10.1111/cgf.14739 1

[10] G. Gomez-Nogales, M. Prieto-Martin, C. Romero, M. Comino-Trinidad, P. Ramon-Prieto, A.-H. Olivier, L. Hoyet, M. Otaduy, J. Pettre, and D. Casas. Resolving collisions in dense 3d crowd animations. *ACM Trans. Graph.*, 43(5), Sept. 2024. doi: 10.1145/3687266 2

[11] D. Holden. Character control with neural networks and machine learning. *Proc. of GDC 2018*, 1:2, 2018. 1

[12] D. Holden, O. Kanoun, M. Perepichka, and T. Popa. Learned motion matching. *ACM Trans. Graph.*, 39(4), aug 2020. doi: 10.1145/3386569.3392440 1

[13] L. Hoyet, A.-H. Olivier, R. Kulpa, and J. Pettré. Perceptual effect of shoulder motions on crowd animations. *ACM Trans. Graph.*, 35(4), July 2016. doi: 10.1145/2897824.2925931 1

[14] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange. Unity: A general platform for intelligent agents, 2020. 3

[15] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017. 3

[16] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2022. 3

[17] A. Kwiatkowski, E. Alvarado, V. Kalogeiton, C. K. Liu, J. Pettré, M. van de Panne, and M. P. Cani. A survey on reinforcement learning methods in character animation. *Computer Graphics Forum*, 41(2):613–639, 2022. doi: 10.1111/cgf.14504 1

[18] A. Kwiatkowski, V. Kalogeiton, J. Pettré, and M.-P. Cani. Understanding reinforcement learned crowds. *Computers Graphics*, 110:28–37, 2023. doi: 10.1016/j.cag.2022.11.007 1

[19] T. Kwon, Y. Lee, and M. Van De Panne. Fast and flexible multilegged locomotion using learned centroidal dynamics. *ACM Trans. Graph.*, 39(4), aug 2020. doi: 10.1145/3386569.3392432 2

[20] K. Lee, S. Lee, and J. Lee. Interactive character animation by learning multi-objective control. *ACM Trans. Graph.*, 37(6), dec 2018. doi: 10.1145/3272127.3275071 1

[21] H. Y. Ling, F. Zinno, G. Cheng, and M. Van De Panne. Character controllers using motion vaes. *ACM Trans. Graph.*, 39(4), aug 2020. doi: 10.1145/3386569.3392422 1, 2, 3

[22] Q. Men, H. P. Shum, E. S. Ho, and H. Leung. Gan-based reactive motion synthesis with class-aware discriminators for human–human interaction. *Computers Graphics*, 102:634–645, 2022. doi: 10.1016/j.cag.2021.09.014 1

[23] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. *PyTorch: an imperative style, high-performance deep learning library*. Curran Associates Inc., Red Hook, NY, USA, 2019. 3

[24] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne. Deeploco: dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Trans. Graph.*, 36(4), jul 2017. doi: 10.1145/3072959.3073602 2

[25] X. B. Peng, Y. Guo, L. Halper, S. Levine, and S. Fidler. Ase: large-scale reusable adversarial skill embeddings for physically simulated characters. *ACM Trans. Graph.*, 41(4), jul 2022. doi: 10.1145/3528223.3530110 2

[26] D. Rempe, Z. Luo, X. B. Peng, Y. Yuan, K. Kitani, K. Kreis, S. Fidler, and O. Litany. Trace and pace: Controllable pedestrian animation via guided trajectory diffusion. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13756–13766, 2023. doi: 10.1109/CVPR52729.2023.01322 1

[27] S. Starke, Y. Zhao, T. Komura, and K. Zaman. Local motion phases for learning multi-contact character movements. *ACM Trans. Graph.*, 39(4), aug 2020. doi: 10.1145/3386569.3392450 1

[28] S. Starke, Y. Zhao, F. Zinno, and T. Komura. Neural animation layering for synthesizing martial arts movements. *ACM Trans. Graph.*, 40(4), jul 2021. doi: 10.1145/3450626.3459881 1, 2

[29] J. Wang, H. Xu, M. Narasimhan, and X. Wang. Multi-person 3d motion prediction with multi-range transformers. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21. Curran Associates Inc., Red Hook, NY, USA, 2024. 1

[30] Y. Wu and K. He. Group Normalization. *International Journal of Computer Vision*, 128(3):742–755, Mar. 2020. doi: 10.1007/s11263-019-01198-w 3

[31] S. Yang, T. Li, X. Gong, B. Peng, and J. Hu. A review on crowd simulation and modeling. *Graphical Models*, 111:101081, 2020. doi: 10.1016/j.gmod.2020.101081 1

[32] H. Zhang, S. Starke, T. Komura, and J. Saito. Mode-adaptive neural networks for quadruped motion control. *ACM Trans. Graph.*, 37(4), jul 2018. doi: 10.1145/3197517.3201366 3