한국컴퓨터그래픽스학회
Korea Computer Graphics Society Vol. 29, No. 4, P. 7~16

캐릭터 복싱 과제에서 GAN 기반 접근법과 강화학습의 효과성 탐구

손서영O 권태수*
한양대학교 일반대학원 컴퓨터소프트웨어학과
ameliacode@hanyang.ac.kr, taesoobear@gmail.com

Exploring the Effectiveness of GAN-based Approach and Reinforcement Learning in Character Boxing Task

Seoyoung SonO Taesoo Kwon*
Department of Computer Science, Hanyang University Graduate School, South Korea

Abstract

For decades, creating a desired locomotive motion in a goal-oriented manner has been a challenge in character animation. Data-driven methods using generative models have demonstrated efficient ways of predicting long sequences of motions without the need for explicit conditioning. While these methods produce high-quality long-term motions, they can be limited when it comes to synthesizing motion for challenging novel scenarios, such as punching a random target. A state-of-the-art solution to overcome this limitation is by using a GAN Discriminator to imitate motion data clips and incorporating reinforcement learning to compose goal-oriented motions. In this paper, our research aims to create characters performing combat sports such as boxing, using a novel reward design in conjunction with existing GAN-based approaches. We experimentally demonstrate that both the Adversarial Motion Prior [3] and Adversarial Skill Embeddings [4] methods are capable of generating viable motions for a character punching a random target, even in the absence of mocap data that specifically captures the transition between punching and locomotion. Also, with a single learned policy, multiple task controllers can be constructed through the TimeChamber framework.

요 약

캐릭터 애니메이션 분야에서 목표 지향적 이동을 위해 원하는 궤적을 재현하는 것은 항상 어려운 과제이다. 생성 모델을 사용하는 데이터 기반 방법은 명시적인 조건 없이 긴 동작 시퀀스를 예측하는 효율적인 방법 중 하나이다. 이러한 방법은 고품질의 결과물을 생성해내지만, 멀리 있는 목표물을 무작위로 타격하는 것처럼 더 어려운 상황의 모션을 합성(synthesis)에 있어서는 제한될 수 있다. 하지만 이는 모션 데이터 클립을 모방하는 GAN Discriminator 를 사용하고 강화학습을 통해 해결할 수 있다. 본 연구는 캐릭터들이 GAN 기반 접근법과 리워드 설계를 통해 복싱을 구현하는 것을 목표로 한다. 논문에서 사용된 두 가지의 최신 연구인 Adversarial Motion Prior 와 Adversarial Skill Embedding 에 대해 비교 실험하며, 또한 복싱을 경쟁 스포츠에 적용하기 위하여 멀티 에이전트 강화 학습을 위한 대규모 self-play 프레임워크인 TimeChamber 를 활용한다.

키워드: 캐릭터 애니메이션, 물리기반 시뮬레이션, 강화학습, GAN
Keywords: Character Animation, Physics-based Simulation, Reinforcement Learning, GAN

1. Introduction

With the advancements in GPUs and the increasing performance of computers, users nowadays can experience diverse virtual environments, plunging themselves into virtual characters. It has become essential that characters move similarly to reality in virtual space. Given a limited amount of motion capture data, developing control strategies for characters has been a long-standing problem in character animation. Traditionally, motion generation has been solved by manually re-arranging motion clips or connecting motion data as nodes via directed graph structure [1], and for generating goal-

*corresponding author: Taesoo Kwon / Department of Computer Science, Hanyang University Graduate School (taesoobear@gmail.com)

Received : 2023.06.28./ Review completed : 1st 2023.07.30. / Accepted : 2023.08.16.
DOI : 10.15701/kcgs.2023.29.4.7
ISSN : 1975-7883(Print)/2383-529X(Online)

directed motion, path planning for a character's valid trajectory, so-called motion planning is an option. However, conventional methods are limited in that they are not capable of producing diverse, complex motions in a new environment that is different from the captured environment.

Recent research interests have shifted to implementing deep learning-based methods and have demonstrated efficient methods by adapting generative models to predict plausible, diverse motion, and reinforcement learning to produce task-based locomotion. Imitation learning is one of the examples of deep learning-based approaches. Multiple studies have demonstrated the application of imitation learning to replicate physics-based motions by mimicking the provided motion database. However, it is limited in that it cannot respond to various scenarios when exposed to adversarial surroundings. To cooperate interactions without explicit programming, Generative Adversarial Imitation Learning [2] is an alternative to imitation learning. Adversarial Motion Prior [3] and Adversarial Skill Embeddings [4] are two notable approaches that apply generative adversarial imitation learning to the field of character animation and have shown remarkable results in generating natural motions. From the perspective of a generative model, these two models leverage Generative Adversarial Networks to provide information on the fake distribution. Compared to approaches such as Variational Autoencoder, this enables the seamless and natural execution of different motions. Based on the data-driven methods, Reinforcement Learning is integrated to enable characters to perform each task while simultaneously preserving the distinctive features of the mocap data. Chapter 2 of this paper will provide an overall survey of deep learning methods.

On top of these state-of-the-art methods, we have implemented the downstream tasks of punching and boxing. Our contributions can be summarized as follows:

- We first review several generative model approaches and reinforcement learning methods for generating physics-based tasks. These methods train the system to perform desired tasks by leveraging embedded skills or motion clips.

- We aim to enable our character to perform punching and boxing tasks, and compare two existing, latest models for generating dynamic motions: Adversarial Motion Prior, and Adversarial Skill Embedding (ASE). Also, we implemented a boxing task with ASE based on Competitive Multi-Agent Reinforcement Learning using the TimeChamber framework. Our redesigned reward functions are shown to be effective in several experiments.

2. Related works

In general, character animation can be categorized based on its attributes: kinematic method and physics-based method. the former predicts action spaces that consist of kinematic poses, while the latter aims to produce joint torques through physics simulation. For the past few years, recent studies have demonstrated generating kinematic or physics-based motion through neural networks. Generating kinematic motions through this approach has demonstrated plausible results without the use of physical formulas [5-7], while a physics-based approach has been shown to create more natural motion [8, 9].

Reconstructing motion through conditioning from latent space or motion manifolds is one of the popular data-driven methods in the latest works. It is meaningful in that its non-linear deformation represents hidden variables or intrinsic features, underlying the data [10]. Generative models are used to produce new variations and transitions of a given character motion by learning a compact latent space representation of motion data and sampling from it.

2.1 GAN in Character Animation

Data-driven methods using generative models such as Convolutional Autoencoders [11], Conditional Variational Encoder (CVAE) [12-14], and Normalizing Flows [15] demonstrated efficient ways of predicting long sequences of motions Though these methods produce long-term motion, several limitations still exist. As CVAE highly depends on the prior distribution of its input data, it may be difficult to establish a connection between different motions, which is known as posterior collapse. This limitation can arise when synthesizing motions in more complex scenarios, such as punching a random target. To solve this, the explicit condition was used additionally to each encoder and decoder to integrate acyclic motions [12] or preprocess motion dataset via motion graph to overcome the lack of transition or connectivity of each dataset [13]. Due to the lack of connection between each corresponding skill and latent space, the discriminator was additionally used in mapping conditioned by state-conditioned prior and posterior prior [14]. This indicates an independent CVAE highly depends on the quantity and distribution of the dataset, which in turn, may fail to create transitions in distinct motions.

Generally, GAN generates new data samples that closely resemble the ground truth by training a generator. This is achieved by training a

discriminator to the point where it cannot distinguish between the ground truth and the generated data. Compared to VAEs, GAN generates diverse, high-quality motions, and it is suitable for transitioning between distinct actions such as kicking and punching. Inspired by Generative Adversarial Imitation Learning (GAIL)[2] framework and motion prior, which was used in pose estimation to indicate the similarity between generated motion and ground truth, Adversarial Motion Prior (AMP)[3] was introduced to generate novel motion sequences without the need of explicit information on selecting and sequencing. Discriminator, which is depicted as AMP, first collects trajectories with policy. Unlike the state-action pair used in GAIL, AMP is trained with state transitions as states are only observed in data. Given trajectory information, the discriminator is trained as a policy by least-squares regression problem with style-reward function. The reward is then recorded trajectory and stored in a replay buffer, which prevents the discriminator from overfitting and stabilizes the training process. Once all trajectories are recorded with rewards, recorded rewards are used to update policy and value functions. The agent here, learns its actions by maximizing the expectation value of the likelihood of the trajectory within a given goal. Finally, the discriminator is updated by using mini batches of transitions from sampled ground truth data and transitions from the replay buffer.

Similar to AMP, Adversarial Skill Embedding (ASE)[4] also adapted discriminator. However, unlike AMP, ASE proposed two stages: a pre-training stage for low-level policy which is conditioned by state and embedded skills, and a transferring embedded skills stage for high-level policy. This hierarchical structure enables policy to learn adequate physics-based actions without learning from scratch, as AMP requires a whole dataset for different policies of each task. In low-level policy training, the policy $\pi(a_t|s_t, z)$ learns skill embedding which is a result from imitating ground truth motions. In this work, the latent space Z is modeled as a hypersphere. As the sphere itself here is expected to have a uniform prior distribution, it allows to prevent the result of unnatural action spaces. In each iteration, a batch of trajectories is collected with a policy on sampled latent from the unit sphere space. Rewards are specified as the sum of a least-squares regression and latent representation from the encoder at each time step. Each process is stored in a data buffer, and later this information will be used in updating the encoder and discriminator by sampling mini batches of transitions. Other works of recent studies have shown generating novel data by using natural language processing. Inspired from ASE, PADL [16] provides an effective interface to users without requiring pre-knowledge, and enables the character to direct its behavior

2.2 Reinforcement Learning

Controlling characters to a purposeful motion is one of the challenges in computer animation. Planning trajectories for characters [17] or supervising learning methods [18] showed promising results in kinematic animation. In physics-based animation, reinforcement learning is widely used for its pure objective [9], imitating reference data [8], or synthesizing motion while optimizing [19]. This is because motions from dynamic models go through a simulator in one way, where non-differentiable prediction cannot be reused as optimization [20].

Multi-agent reinforcement learning (MARL) involves developing algorithms for multiple agents to learn and interact with one another in a shared environment. Each autonomous agent receives rewards by their individual actions and collective states from a multi-agent environment while cooperating or competing with others. Several MARL works have demonstrated competitive policies for physics-based character control [21], as well as for hide-and-seek games [22]. For a human-like result, Won et al. [23] first, imitates the motion and then, train the two agents in a competitive scenario. Similar to this approach, TimeChamber [24], a self-play framework for multiple agents inspired by ASE, trains two agents to discover skills and is constrained by low-level policy, where it is trained previously in the pre-training stage of ASE. The overall transferring stage is based on a classic PPO self-play algorithm, which allows multiple agents to learn from playing against each other and improve while training.

3. Experiments

3.1 Data preprocessing

First, we collected related mocap data from CMU [25] for our task. Our database included from simple locomotion to boxing actions. To adjust the existing humanoid model, the mocap dataset was retargeted. Irrelevant joints were dropped in this process while keeping similar joints by their names. As a result, 31 joints in the original data were reduced to 28 joints. Inspired by Won et al. [23] both hands in humanoid are 1.75 times larger (see Figure 1).

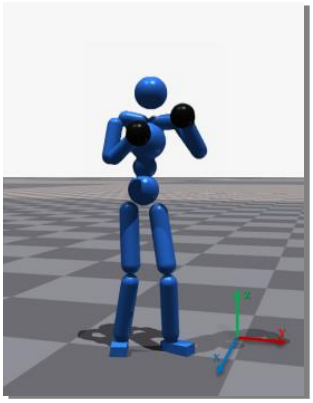


Figure 1. Humanoid character heading x-axis in default pose

Then, the data were weighted according to the different motion types of files for balancing overall distribution. The weights were set according to the ratio based on the largest amount of motion type. Suppose there are two data types: A and B, and type A is larger than B. To apply such a rule, the following weight would be:

$$weight_b = \frac{n(A)}{n(B)} \quad (1)$$

Table 1 summarizes the type of motion data and weights that were used in training. The same datasets were used in each task.

Dataset	Clips	Length (secs)	Weights
Locomotion	17	248.46s	1.0
Boxing	5	74.11s	3.4

Table 1. Summary of applied weights in each data

3.2 Character Punching Task

Character punching tasks consist of two independent scenarios: run straight forward then, punch (punching task), or run towards to target while facing randomly (joystick task). Both scenarios use the same mocap data and the same environment. The character will be directed to attack a plain cuboid target that measures 0.4m x 0.4m x 1.8m. The target will be placed randomly within a range of 5 to 10m.

Using Motion Prior, we have designed a new environment for characters to randomly punch targets. Unlike previous work on AMP using Bullet physics engine [26], we conducted experiments utilizing the publicly accessible AMP implementation built upon Isaac Gym [27], a GPU-based physics simulated environment, to simulate in 4096 parallel environments. A single NVIDIA RTX 2080Ti GPU was used in this case. We also took advantage of ASE to compare AMP. The same motion assets and environment used in AMP were used in training low-level policy in ASE. Unlike the AMP framework, ASE employs a low-level policy in transferring stage. High-level policy in this stage trains a meaningful latent vector based on the state, goal-state, and reward. Each training time took approximately 10 days in pre-training and 22 hours in transferring. Both stages were trained on a single GPU, NVIDIA RTX 2080Ti GPU.

Inspired by the task weighting approach in PADL[16], we departed from the use of static weights for both the task and the discriminator. Instead, we adopted dynamic task weights on training ASE, akin to those regulated by a proportional-derivative (PD) controller. This adaptive scheme facilitates a balanced consideration of task and skill rewards promoting efficiency.

3.2.1 Observation Spaces

Our observation spaces for two different scenarios are similar to each other. We additionally appended local target moving and facing direction in the joystick task. The target moving direction in the joystick task is a local direction between the target cuboid and the character's initial position. The target facing direction, on the other hand, is a random local facing direction where it remains until the environment resets. Both directions are projected onto the ground. Table 2 describes our continuous observation spaces used in detail.

		Index	Description
A	B	0-2	Local position between the target cuboid and the character's root
		3-6	Local rotation between the target cuboid and the character's root
		7-10	Local linear velocity between target linear velocity and the character's root
		11-14	Local angle velocity between target angle velocity and the character's root
		15-16	Local target direction between the target moving direction and the character's root
		17-18	Local facing direction between target facing direction of the character 's root

Table 2. Observation spaces in two downstream tasks: A-joystick task, B-punching task

3.2.2 Task-Related Reward Function

To strike a target in a natural way, we have divided our task into a 2-step behavior as suggested by AMP. If the target's location and root position's difference is longer than 1.2m, the character first approaches the given target. The character then strikes the target if it is within the distance threshold. When punching a target, we have mainly focused on the target's up-vector solely whether it is struck or

not. Based on AMP striking downstream task's reward design, we redefined and improved r_{far} to reflect all the circumstances of relative position, velocity, and facing term of the character. These terms are multiplied as a product in r_{far}. Also, compared to the previous return, the facing term is additionally appended. Our task-related reward function is defined as follows:

$$r_t = \begin{cases} 1.4 & \text{if target hits} \\ 0.3\, r_{near} + 0.3 & \text{if } \| \, pos_{target} - pos_{root} \, \| < 1.2m \\ 0.3\, r_{far} & \text{else} \end{cases}$$

In r_{near}, the reward motivates the character's hands to reach and strike a target. The first term r_{dist} encourages the character to punch the target by hand while r_{vel} constrains glove's striking velocity. The largest r_{near} is then, selected between the left and right gloves. d here, is a unit vector of character facing the direction to a goal.

$$r_{near} = 0.2\, r_{dist} + 0.8\, r_{vel} \quad (2)$$

$$r_{dist} = \exp\left(-4 \, \| \, pos_{target} - pos_{hand} \, \|^2\right) \quad (3)$$

$$r_{vel} = \text{clamp}(d \cdot vel_{hand}, 0.0, 1.0) \quad (4)$$

On the other hand, r_{far} stimulates the character to get closer to the target. Both the punching task and the joystick task share the same reward function. Overall, r_{far} aims to maximize and confine its value to 1.0. Equivalent to the first term in r_{near}, r_{dist} in r_{far} enables character to get closer to target.

$$r_{far} = r_{dist} * r_{vel} * r_{facing} \quad (5)$$

$$r_{dist} = \exp(-\| \, pos_{target} - pos_{root} \, \|^2) \quad (6)$$

The second term r_{vel} diminishes the error between the character's velocity vel_{root} with goal direction $d^*$and target speed vel^* (1.2m/s). err_{vel} calculates tangential speed error where vel_{root} is a root velocity projected onto the ground.

$$err_{speed} = vel^* - d^* \cdot vel_{root} \quad (7)$$

$$err_{vel} = \sum (vel_{root} - err_{speed} * d^*) \quad (8)$$

$$r_{vel} = \exp\left(-4 \, \| \, (err_{vel})^2 + (err_{speed})^2 \, \|\right) \quad (9)$$

To directly face the desired direction, we added the facing reward r_{facing}. r_{facing} allows the character to minimize the error between the target facing direction $\hat{d}$ and the current character's heading direction $\bar{d}$. Both heading directions are normalized. Ablation study for facing reward, see Figure 3.

$$r_{facing} = \exp(-5 \, \| \, \bar{d} \cdot \hat{d} - 1 \, \|) \quad (10)$$

In short, r_{far} maximizes the overall reward value by minimizing the distance between the character and the target (r_{dist}), closely approximating the specified speed and the linear velocity (r_{vel}) and ensuring the character faces the target (r_{facing}). By satisfying all these conditions, the character is prompted to punch the given cuboid.

3.3 Character Boxing Task

In this task, we have applied a framework for multi-agents to compete with each other as boxing, TimeChamber. Based on ASE, the competitive policy learned by the discriminator will be transferred as a high-level policy to each agent. In this way, both agents learn tactics against their opponents based on basic skills. In addition to the previous task, our boxing task was trained in Isaac Gym, with 4096 parallel environments in a single GPU. Motivated by ASE task training, TimeChamber learns adequate latent information which fits the situation. For diversity in playing boxing games, TimeChamber has implemented a pool of opponent players where it is prioritized and sorted by winning rate and consists of high-level policy, winning rate, and environment index. Through the training, opponents are sampled based on their winning rate, then allocated to each parallel environment. End of each game, the winning rates are updated in the player pool. This depends on the ELO rating system, which calculates the relative skill levels of two players.

Compared to a single agent environment, observation space in a competitive environment additionally contains the opponent's behavior state in local. The observation size is 65 in total.

3.3.1 Task-Related Reward Function

Our boxing strategy is akin to that of a punching task. When the opponent is considered far from the player, the player approaches the opponent player. Then, the player maneuvers to strike down the opponent with a high return.

$$r_t = \begin{cases} 500 & \text{if the opponent falls} \\ r_{near} & \text{if } \| \, pos_{target} - pos_{root} \, \| < 1.75m \\ r_{far} & \text{else} \end{cases}$$

When approaching each other, r_{near} enables the character to get closer to the opponent (r_{pos}) while maintaining a target speed 1.2m/s (r_{vel}). r_{facing} allows players to face forward since attacking an opponent from behind is considered as violation in boxing (see Figure

9). This minimizes the error between target facing direction $\hat{d}$ and the current character's heading direction $\bar{d}$.

$$r_{far} = 0.5r_{pos} + 0.4r_{vel} + 0.1r_{facing} \quad (11)$$

$$r_{pos} = \exp\left(-0.5 \parallel pos_{op} - pos_{root} \parallel^2\right) \quad (12)$$

$$r_{vel} = \exp\left(-4.0 \parallel \max(0, 1.2 - d \cdot vel_{root}) \parallel^2\right) \quad (13)$$

$$r_{facing} = \min\left(0.0, \bar{d} \cdot \hat{d}\right) \quad (14)$$

In r_{near}, unlike the punching task, a penalty is added if the player itself has fallen or crossed the arena boundary. A penalty for facing direction has also been added as a regulation in boxing. Without a facing penalty, the player may tend to attack their opponent while holding down from behind. If the opponent falls, the player gets a reward value of 500. The overall reward function is similar to that of downstream strike task's from TimeChamber. Table 3 includes the weights that were used in the reward function.

$$r_{near} = w_{damage} * r_{damage} + w_{close} * r_{close} + w_{facing} * r_{facing} + w_{fall} * r_{fall} - w_{energy} * r_{energy} - w_{penalty} * r_{penalty}$$

r_{damage} measures how player have damaged by force. All forces are normal forces that are contacted to the player ($F_{op \to ego}$) or the opponent $F_{ego \to op}$).

$$r_{damage} = \min\left(-200, F_{ego \to op} - 2 * F_{op \to ego}\right) \quad (15)$$

r_{close} motivates the character to punch the opponent's target point: torso and head. r_{close} is identical to r_{near} that was used in the punching task. r_{facing} encourages players and the opponent to face each other. d denotes the opponent's local facing direction to the player while $\hat{d}$ is the character's local facing direction to the opponent. Both directions are normalized. The previous facing reward from Won et al. [23] presents a challenge in enabling characters to perform proper boxing maneuvers. It only considers whether the character faces the target, disregarding the opponent's heading direction. This allows the agent to attack in the opposite direction (see Figure 9).

$$r_{facing} = \exp\left(-5 \parallel -\bar{d} \cdot \hat{d} - 1 \parallel\right) \quad (16)$$

If the opponent is about to fall, r_{fall} calculates the error between the opponent's up-vector v_{op} and global up-vector v_{up}. To restrain character to attack backward we only give fall reward when facing each other. r_{energy} induces characters to behave effectively and less aggressively where a_{joint} is the angular acceleration of character's joint and l is a distance between the players.

$$r_{fall} = 0.6 * \min\left(1 - v_{up} \cdot v_{op}, 0.0\right) \quad (17)$$

$$k_{dist} = \exp\left(-10 * \sum || \max(0.0, l - 1.0)||^2\right) \quad (18)$$

$$r_{energy} = k_{dist} * \sum \left|\left|a_{joint}\right|\right|^2 \quad (19)$$

w_{damage}	1.0
w_{close}	4.0
w_{facing}	10.0
w_{fall}	200.0
w_{energy}	0.001
$w_{penalty}$	30.0

Table 3. Weights in boxing reward function

4. Result

4.1 Evaluation of Character Punching Task

Compared to the previous AMP environment, it required approximately 8 hours which resulted in 140 hours, where the training time has been significantly reduced. This can be done by training with parallel environments, which allows the network to collect data faster. Not only does this improves the training time, but it also improves the overall correlation between multiple trajectories in the dataset. Our work snapshots are depicted in Figure 2.

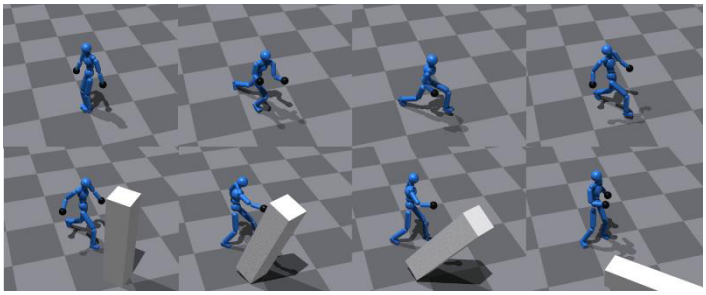


Figure 2. Based on generated motions from control policy, the character heads toward to target and strikes it.

If the facing term is ignored, the character tends to punch a target disregarding the facing direction, which leads to an unnatural style of locomotion. Without the term, the character maintains the initial facing direction while approaching (see Figure 3). Throughout the experiment, having a reward for facing direction is significant as it allows the character to retain its initial random facing while moving toward the target in the joystick task (see Figure 4). Although tuning the facing error is necessary, it still holds importance. Overall learning curves increase gradually in both tasks.

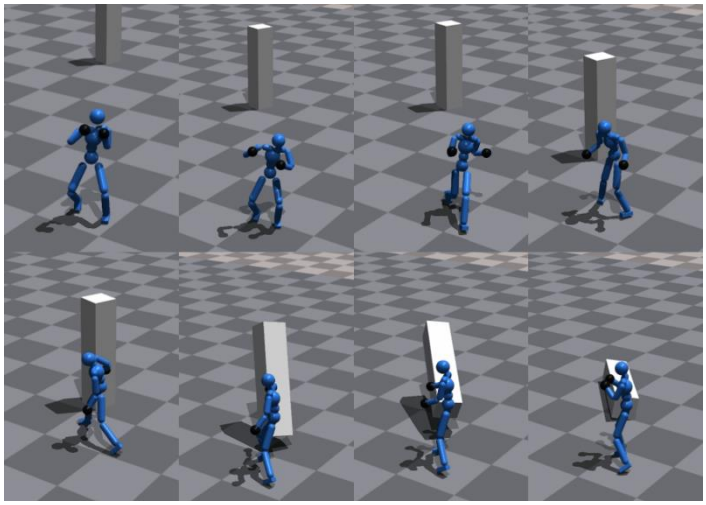


Figure 3. The character's initial heading direction is maintained while approaching the target, resulting in unnatural locomotion.

In the joystick task, the character manages to face the desired direction while heading towards to the target. The red arrow and blue arrow are illustrated in Figure 4 where each arrow points to the target and desired heading direction respectively.

Both tasks' learning curves increased gradually in training. This denotes that a character tends to behave to obtain maximum rewards. Figure 5 shows the learning curves of mean rewards on each task.

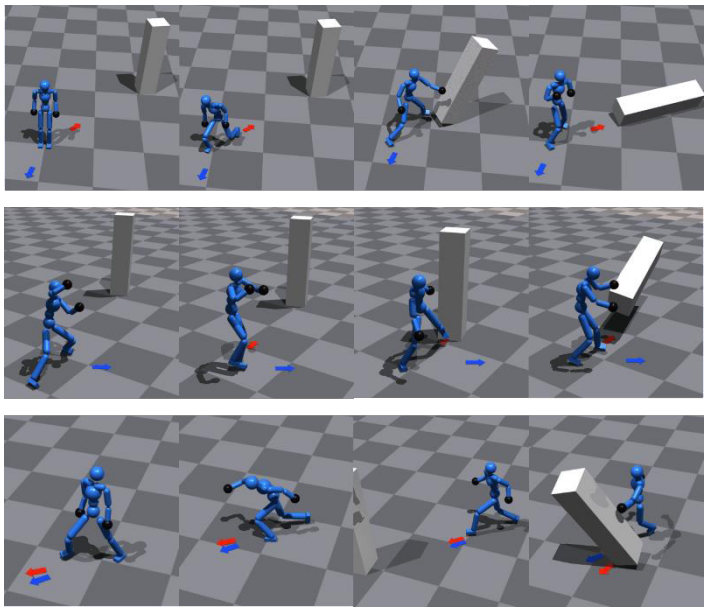
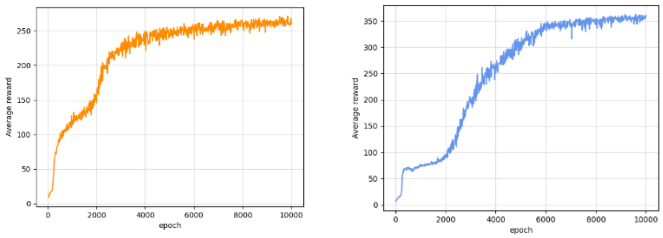


Figure 4. Characters tend to face the desired random direction which is represented in a blue arrow. (Top) Facing Backwards, (Middle) Facing Sideways, (Bottom) Facing Front.



(a) Punching (b) Joystick

Figure 5. Learning curves of mean reward on each task

Our experiment with ASE on punching task fulfills its task too. However, while the character moves toward the target, it behaves unnaturally compared to the result from AMP. This may have occurred due to the exploitation of a discriminator, which results in the character to behave a subset of specific skills as the given latent information returns the utmost reward (see Figure 6). Furthermore, the data utilized in the research only included essential information due to limitations in available GPU resources. To address this issue potential strategies involve training a low-level policy utilizing a range of diverse datasets, or alternatively, generating distinct latent vectors for each individual dataset. These methods aim to effectively mitigate the phenomenon of mode collapse. Overall training time took approximately 22 hours. Figure 7 shows the learning curves of its average task returns and task weight. Task weight is updated automatically via PD Controller. This enables to balance between task and style reward weight.

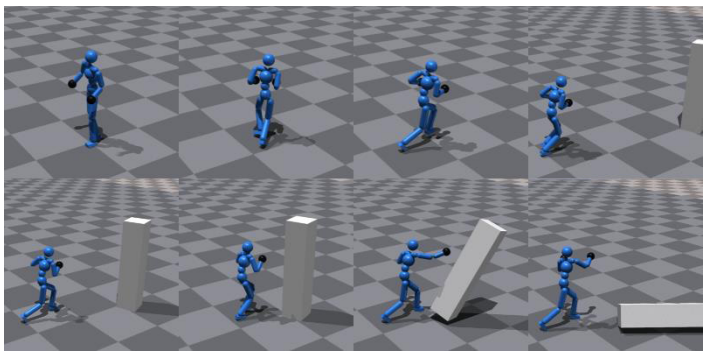
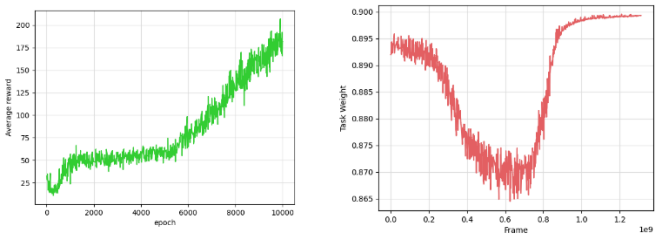


Figure 6. The humanoid faces toward to target and strikes it. Unlike the same task trained from AMP, the character tends to punch less aggressively with low energy.



(a) Mean reward (b) Task weight

Figure 7. Learning curves of average task return in punching task via ASE and its task weight.

4.2 Evaluation of Character Boxing Task

Training time for character boxing task took around 6 to 7 days. Multi-agent reinforcement learning spent a longer time due to its non-stationery feature and increased complexity. Also balance between exploration and exploitation trade-off might be a reason for this as well. Hence, learning curves in MARL might not increase constantly, especially in scenarios like competitive, zero-sum game environments. Meaningful skills emerged as the epoch increased. After training with 40000 epochs, the characters attacked and defended themselves naturally (see Figure 8).

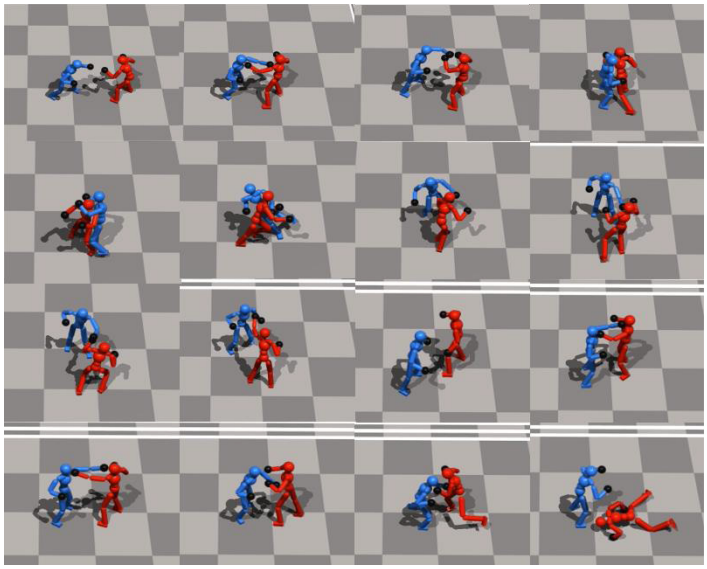


Figure 8. Characters tend to stretch their arms to defend themselves from their opponent. Also, the character learns to headlock the opponent and it defends itself by untangling the tactic.

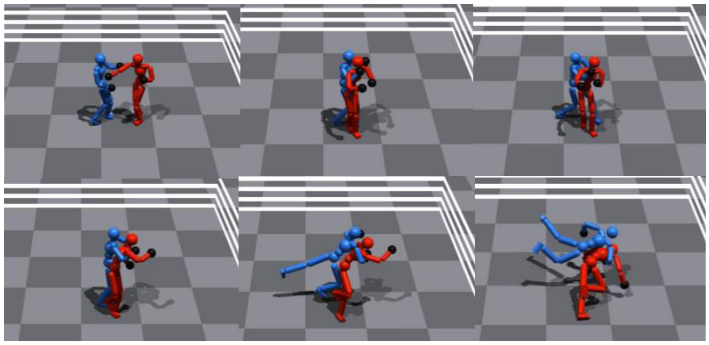


Figure 9. Won et al. [23]'s prior reward method poses a challenge for characters attacking from behind, as it ignores the opponent's heading direction and may cause the agent to attack in the wrong direction.

5. Conclusion

In this paper, we explored effective frameworks of recent studies with our character punching and boxing tasks by comparing two existing methods: AMP and ASE. Without requiring explicit conditioning in the form of sequences or labels, both models illustrated natural transitions on locomotion and punching.

However, AMP is limited to dealing with various tasks with a single motion prior. Since a discriminator alone does not contain implicit information in the latent space, which enables to recover distinct behavior for each data, it becomes necessary to train it from scratch for different individual tasks. Also, AMP is prone to mode collapse if the task objective is vague, which leads to repeating specific behavior. ASE, on the other hand, uses a low-level policy to learn a latent space of meaningful skills. During the pre-training task, the low-level policy is trained to balance and imitate behaviors from the dataset. When training task-oriented high-level policy, the low-level policy constrains the set of possible actions. This reduces exploiting abnormal movements compared to the prior model. Still, ASE is also prone to mode collapse in that the combined reward between the latent information of the corresponding trajectory and discriminator may exploit only a portion of the skills. PADL solves it by allocating unique latent information to each motion dataset. Motivated by the task weight from PADL, instead of using fixed weights for tasks and the discriminator, we implemented task weights which are adjusted in a way similar to a PD controller.

We also demonstrate a competitive boxing task performed by two characters via TimeChamber, an ASE-based framework. Due to the complex nature of competitive MARL, the training time was longer compared to the single-agent environment. Also, finding the balance between exploration and exploitation is intricate as the agents observe

moving objects while maximizing their return. As the training epoch increases, the character becomes capable of performing more meaningful and sophisticated skills. However, as noted in ASE, some movements such as “walking” towards the opponent aren’t natural.

For future work, implementing a diffusion model could be a promising direction. The diffusion model is one of the popular methods in recent works and it generates high-quality kinematic motions through stochastic processes [28-30]. Physics-based motion controllers based on a pre-trained diffusion model also exists [31]. However, most of the state-of-the-art existing diffusion models aim to control a single character. It would be intriguing to build an adversarial environment to interact with multiple agents while the user uses natural language prompts to control characters. We look forward to building interesting environments on top of generative models.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (NRF-2020R1A2C1012847).

References

[1] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," *ACM SIGGRAPH 2008 classes*, pp. 1-10, 2008.

[2] J. Ho, and S. Ermon, “Generative adversarial imitation learning,” *Advances in neural information processing systems,* vol. 29, 2016.

[3] X. B. Peng *et al.*, “Amp: Adversarial motion priors for stylized physics-based character control,” *ACM Transactions on Graphics (TOG),* vol. 40, no. 4, pp. 1-20, 2021.

[4] X. B. Peng *et al.*, “ASE: Large-Scale Reusable Adversarial Skill Embeddings for Physically Simulated Characters,” *arXiv preprint arXiv:2205.01906*, 2022.

[5] D. Holden, T. Komura, and J. Saito, “Phase-functioned neural networks for character control,” *ACM Transactions on Graphics (TOG),* vol. 36, no. 4, pp. 1-13, 2017.

[6] H. Zhang *et al.*, “Mode-adaptive neural networks for quadruped motion control,” *ACM Transactions on Graphics (TOG),* vol. 37, no. 4, pp. 1-11, 2018.

[7] D. Holden *et al.*, “Learned motion matching,” *ACM Transactions on Graphics (TOG),* vol. 39, no. 4, pp. 53: 1-53: 12, 2020.

[8] X. B. Peng *et al.*, “Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning,” *ACM Transactions on Graphics (TOG),* vol. 36, no. 4, pp. 1-13, 2017.

[9] W. Yu, G. Turk, and C. K. Liu, “Learning symmetric and low-energy locomotion,” *ACM Transactions on Graphics (TOG),* vol. 37, no. 4, pp. 1-12, 2018.

[10] A. Elgammal, and C.-S. Lee, "The role of manifold learning in human motion analysis," *Human Motion*, pp. 25-56: Springer, 2008.

[11] D. Holden, J. Saito, and T. Komura, “A deep learning framework for character motion synthesis and editing,” *ACM Transactions on Graphics (TOG),* vol. 35, no. 4, pp. 1-11, 2016.

[12] H. Y. Ling *et al.*, “Character controllers using motion vaes,” *ACM Transactions on Graphics (TOG),* vol. 39, no. 4, pp. 40: 1-40: 12, 2020.

[13] J. Won, D. Gopinath, and J. Hodgins, “Physics-based character controllers using conditional VAEs,” *ACM Transactions on Graphics (TOG),* vol. 41, no. 4, pp. 1-12, 2022.

[14] H. Yao *et al.*, “ControlVAE: Model-Based Learning of Generative Controllers for Physics-Based Characters,” *ACM Transactions on Graphics (TOG),* vol. 41, no. 6, pp. 1-16, 2022.

[15] G. E. Henter, S. Alexanderson, and J. Beskow, “Moglow: Probabilistic and controllable motion synthesis using normalising flows,” *ACM Transactions on Graphics (TOG),* vol. 39, no. 6, pp. 1-14, 2020.

[16] J. Juravsky *et al.*, "PADL: Language-Directed Physics-Based Character Control." pp. 1-9.

[17] S. Agrawal, and M. van de Panne, “Task-based locomotion,” *ACM Transactions on Graphics (TOG),* vol. 35, no. 4, pp. 1-11, 2016.

[18] K. Lee, S. Lee, and J. Lee, “Interactive character animation by learning multi-objective control,” *ACM Transactions on Graphics (TOG),* vol. 37, no. 6, pp. 1-10, 2018.

[19] J. Merel *et al.*, “Catch & carry: reusable neural controllers for vision-guided whole-body tasks,” *ACM Transactions on Graphics (TOG),* vol. 39, no. 4, pp. 39: 1-39: 12, 2020.

[20] L. Fussell, K. Bergamin, and D. Holden, “Supertrack: Motion tracking for physically simulated characters using supervised learning,” *ACM Transactions on Graphics (TOG),* vol. 40, no. 6, pp. 1-13, 2021.

[21] T. Bansal *et al.*, “Emergent complexity via multi-agent competition,” *arXiv preprint arXiv:1710.03748*, 2017.

[22] B. Baker *et al.*, “Emergent tool use from multi-agent autocurricula,” *arXiv preprint arXiv:1909.07528*, 2019.

[23] J. Won, D. Gopinath, and J. Hodgins, “Control strategies for physically simulated characters performing two-player competitive sports,” *ACM Transactions on Graphics (TOG),* vol. 40, no. 4, pp. 1-11, 2021.

[24] Z. L. Huang Ziming, Wu Yutong, Flood Sung. "TimeChamber: A Massively Parallel Large Scale Self-Play Framework," https://github.com/inspirai/TimeChamber.

[25] CMU, "CMU Graphics Lab Motion Capture Database," 2002.

[26] E. Coumans, “Bullet physics library,” *Open source: bulletphysics. org,* vol. 15, no. 49, pp. 5, 2013.

[27] V. Makoviychuk *et al.*, “Isaac gym: High performance gpu-based physics simulation for robot learning,” *arXiv preprint arXiv:2108.10470*, 2021.

[28] G. Tevet *et al.*, “Human motion diffusion model,” *arXiv preprint arXiv:2209.14916*, 2022.

[29] M. Zhang *et al.*, “Motiondiffuse: Text-driven human motion generation with diffusion model,” *arXiv preprint arXiv:2208.15001*, 2022.

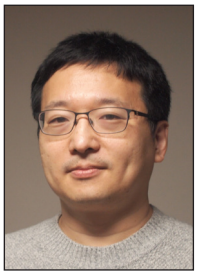
[30] Y. Shafir *et al.*, “Human Motion Diffusion as a Generative Prior,” *arXiv preprint arXiv:2303.01418*, 2023.

[31] Y. Yuan *et al.*, “PhysDiff: Physics-Guided Human Motion Diffusion Model,” *arXiv preprint arXiv:2212.02500*, 2022.

〈 저 자 소 개 〉

손 서 영

- 2017 ~ 2021 세종대학교 소프트웨어학과 학사
- 2021 ~ 2023 한양대학교 컴퓨터소프트웨어학과 석사
- 관심분야: Character animation, Reinforcement learning,
- Physics based simulation, Pose estimation
- https://orcid.org/0009-0009-7291-3953



권 태 수

- 1996-2000 서울대학교 전기컴퓨터공학부 학사
- 2000-2002 서울대학교 전기컴퓨터공학부 석사
- 2002-2007 한국과학기술원 전산학전공 박사
- 관심 분야: Character Animation, Physically-based Animation
- https://orcid.org/0000-0002-9253-2156