Figure Skating Simulation from Video

Ri Yu, Hwangpil Park, and Jehee Lee

Seoul National University, South Korea

Figure 1: Figure skating Sequence. From right to left, a male figure skater performs a three-turn and followed by a salchow jump.

Abstract

Figure skating is one of the most popular ice sports at the Winter Olympic Games. The skaters perform several skating skills to express the beauty of the art on ice. Skating involves moving on ice while wearing skate shoes with thin blades; thus, it requires much practice to skate without losing balance. Moreover, figure skating presents dynamic moves, such as jumping, artistically. Therefore, demonstrating figure skating skills is even more difficult to achieve than basic skating, and professional skaters often fall during Winter Olympic performances. We propose a system to demonstrate figure skating motions with a physically simulated human-like character. We simulate skating motions with non-holonomic constraints, which make the skate blade glide on the ice surface. It is difficult to obtain reference motions from figure skaters because figure skating motions are very fast and dynamic. Instead of using motion capture data, we use key poses extracted from videos on YouTube and complete reference motions using trajectory optimization. We demonstrate figure skating skills, such as crossover, three-turn, and even jump. Finally, we use deep reinforcement learning to generate a robust controller for figure skating skills.

CCS Concepts

- Computing methodologies → Physical simulation; Motion processing; Theory of computation → Reinforcement learning;

1. Introduction

Simulating human locomotion is one of the main topics in the research area of graphics, and many works have been conducted in recent decades. Advances in control methods have allowed us to simulate both simple and dynamic human behaviors as well as interact with complex environments. First, basic human locomotion, such as walking [LKL10] and, running [KH10], was simulated physically. Since then, the simulation of more active movements, such as climbing [KL17, NRH17], riding a bicycle [TGLT14], playing basketball [LH18], and swimming [SLST14], has been researched. Currently, we can not only achieve simulation of complicated motions but we can also simulate interactions with diverse environments, such as uneven terrain, walls, bicycles, and even water. However, locomotion on ice remains a challenge to simulate because there are several constraints, such; balancing on ice, and being non-holonomic. Although Wang et al. [WFH10] did simulate walking motion on ice-like ground by employing a lower friction coefficient to demonstrate the robustness of their controller, research on real skating motion has not been conducted. Recently, Skaterbot [GPD18] was published, which provides designing and optimizing tools for skiing robots, but it uses the wheels as effectors, rather than blades, and it does have an ice environment.

In this paper, we propose a framework for physically simulating figure skating on ice. Skaters wear skate boots and slide forward on a slippery ice surface while manipulating the contact force and balancing. In particular, professional figure skaters can represent dynamic moves, such as jumps, artistically. However, this is difficult even for experts, and Olympic skaters often fall. To control these
movements, we must consider and resolve balancing and contact issues.

To be good at skating, it is essential to be competent with skate blades. The most important factor is understanding the nature of the skate blade and its movement. From contact between the blade and the ice, the skater can take advantage of the low friction and go forward in the direction of the blade. To reflect these characteristics of the blade, we apply a non-holonomic constraint to the system.

One major problem in simulating skating motion is that there are few available skating motion data because skating motion is too dynamic to capture, and the area that must be covered is too broad. Many simulations use motion capture data to mimic human movement, this was not possible for our study. To handle this issue, we generate our own skating motion data. We watched many skating-related videos on the internet and observed skating motions. We found that skating motion is very dynamic, but the deviation between frames is not large. Therefore, we concluded that we can generate motion by selecting key frames well. To obtain key poses, we employed a technique of obtaining a 3D pose from an image [KBJM18].

Even if we obtain key poses from video, they do not have exact global positions, so there is no time and space coherence. Therefore, we apply the trajectory optimization method suggested by Al Borno et al. [ABDLH13] to complete our skating reference motion. By choosing objectives for each key pose properly, key poses are optimized in a way that minimizes the cost function. As a result, we can obtain smooth motions. After generating a reference motion, we have to control the skater model. To find a robust controller for each skating skill, we use reinforcement learning [PALvdP18].

To evaluate our system, we demonstrate several skating motions: crossover, three-turn, and jump. Further, we made a short program using these figure skating skills. To validate the robustness of our controller, we conducted several experiments.

This study has three main contributions. First, to reflect the characteristics of the movement of the skate blade, we implement a skate blade-ice contact model using non-holonomic constraints. Secondly, we succeeded in generating figure skating motions without motion capture data. We used a pose estimation method to obtain key poses from video, which infuse naturalness into the simulation. Then, we can obtain a smooth reference motion using trajectory optimization. Finally, we can simulate figure skating skills physically with a full body character. Thirdly, we generate robust controllers for each figure skating skill using deep reinforcement learning (DRL). Our controller is robust to various speeds and external perturbation.

2. Related Work

Figure skating is very difficult to simulate. For instance, the environment of ice with a very low friction coefficient means that contact between the skate and the ice is very difficult to simulate. Moreover, skating motions are very dynamic and fast, so simulating these motions on ice is a very challenging task.

To date, there have been few attempts to simulate ice skating. In biomechanics, speed skating simulation was conducted with a simple model [vdKVvdHS17, vdKSvdHV18]. Other than this study, most studies on figure skating analyze the force or pose during a jump to determine its effects on injuries in athletes [CMJMS84, KAS93, KH96, HHK06].

The closest study to ours in computer graphics is Skaterbot [GPD*18]. It can design the character structure, and according to that morphology, the motions are optimized. The differences are that they use wheels as end-effectors, and they do not assume the ground is ice. Moreover, their characters are mostly quadrupeds, and no humanoid character is used. We use a full body human character for figure skating simulation with a thin blade on each foot that is in contact with the ice surface.

Our goal is to simulate diverse figure skating skills physically and robustly. In computer graphics, to simulate human movement, we require reference information to follow. Essentially, there are two categories for obtaining reference motion: using motion capture data or not. Motion capture data are obtained from real human actions, so they implicitly contain the criteria of human motor skills and guarantee naturalness. Therefore, a number of studies have used motion capture data to simulate human movement.

At first, walking motion using data-driven control is simulated using 2D characters [SKL07, SvdP05], and then more challenging motions, such as turns, and spins, can be made possible with 3D characters [dSAP08, MLPP09, LKL10]. Lee et al. [LPKL14] tracked motion capture data using quadratic programming with a full body musculoskeletal model. Kwon et al. [KH10] used an inverted Pendulum on a cart model to simulate running. Liu et al. [LH17] used deep Q-learning (DQL) to control each fragment based on the SAMCON algorithm [LYvdP*10] and simulated skateboarding. Also, Liu et al. [LH18] used trajectory optimization and DRL to control the arm motion while playing basketball. Kang et al. [KL17] used motion capture data, but they changed the end-effector positions to contact with the complex environment properly. Peng et al. [PBVYD17, PALvdP18] mimicked motion capture reference data using DRL.

However, there are some difficulties in obtaining motion capture data for figure skating. First of all, to capture skaters, human resources and motion capture camera setup in an ice rink are required, which is costly. Moreover, in the clean-up stage, because figure skating motion is very dynamic and fast, marker occlusion or missing occur frequently. Therefore, we decided not to use motion capture data. There have been previous studies that do not use motion capture data. Without motion capture data, the characteristics of human movement must be determined.

SIMBICON [YLvdP07] can generate human walking motion using key poses. Wang et al. [WHF10] used a SIMBICON-based controller that can deal with uncertainty. Won et al. [WPL18] simulated a dragon because it is impossible to obtain motion capture data for a fictional creature. They used user-provided keyframes for the dragon, and based on them, they built a system that can find a policy to steer or to conduct other skills using DRL.

Al Borno et al. [ABDLH13] successfully generated many motions, including walking and spinning, using simple objectives according to a window optimized using co-variance matrix adaptation evolution strategy (CMA-ES). Climbing has been simulated...
using CMA-ES without the use of motion capture data [NRH17]. Swimming has been reproduced using a CPG-based locomotion controller that can generate muscle contraction signals automatically [SLST14]. Recently, Yu et al. [YTL18] used reinforcement learning to learn basic motor skills, they applied curriculum learning and used symmetry as a reward. Hu et al. [HLL+19] simulated skiing motion given a small set of control inputs by applying skiing techniques.

Recently, Peng et al. [PKM+18] successfully extracted reference motion from videos. They used pose estimation techniques from images [WRKS16,KBJM18]. This inspired our study, where we use figure skating videos on YouTube to obtain key pose data. In this study, we choose key pose images from video and use a framework by Kanazawa et al. [KBJM18] to obtain a 3D character pose. After that, we use a trajectory optimization method proposed by Al Borno et al. [ABDHL13] to generate reference motion.

3. Overview

Our framework have two main steps to simulate the figure skating (Figure 2). When input images from video are given by the user, the first step is a data acquisition process. To obtain the reference data for skating simulation, we extracted key poses from inputs using human mesh recovery (HMR) [KBJM18], a pose estimation technique, which is explained in Section 5. Secondly, We used trajectory optimization to complete the reference motion proposed by Al Borno et al. [ABDHL13]. They generated several forms of human locomotion, such as balancing, walking, spinning, and back-flipping using high-level objectives, and they used CMA-ES [Han06] for trajectory optimization. The inputs of CMA-ES are the key poses and user-specified durations from the data acquisition process. The details are described in Section 6.1.

Through the processes mentioned above, we can simulate individual figure skating skills, such as crossover, three-turn, and jump. To obtain a robust controller for each skill, we apply DRL, which is covered in Section 6.2.

4. Skating Simulation

Our simulation satisfies the equations of motion (Equation 1).

\[
M(q)\ddot{q} + c(q,\dot{q}) + g(q) = \tau + \sum_c J_c f_c, \tag{1}
\]

where \(M\) is the inertia matrix, \(c\) is a matrix of Coriolis and centrifugal forces, and \(g\) is gravitational force. \(q\), \(\dot{q}\), and \(\ddot{q}\) are angle, velocity and acceleration of joint, respectively. \(\tau\) is joint torque, \(J_c\) is the Jacobian matrix for contact, and \(f_c\) is contact force.

To control the simulated character by tracking a key pose sequence or reference motion, we use a proportional-derivative (PD) controller.

\[
\tau = K_p(q_{des} - q) - K_d\dot{q}, \tag{2}
\]

where \(q\) and \(\dot{q}\) are posture and velocity represented in generalized coordinates, respectively, \(q_{des}\) is the pose to track, and \(K_p\) and, \(K_d\) are the proportional and derivative gain matrices, respectively. Because naively choosing gains often leads to unstable simulation, PD gains should be tuned carefully. Therefore, we used a stable proportional-derivative (SPD) controller [TLT11].

In the following subsections, we describe the constraints for skating on ice surface. First, we introduce the non-holonomic constraints restricting the blade’s moving direction (Section 4.1). We implement the non-holonomic constraints using constrained impulse. However, under these constraints, the simulated character can not change the sliding direction unless lifting the leg and changing the direction of the blade. To make the character turn, we relaxed the non-holonomic constraints when tilting (Section 4.2). In this paper, we did not implement the exact figure skating blade design. Instead, we approximated the turning using the relationship of inclined angle and the rotation angle.

4.1. Non-holonomic Constraints

The blade should slide along the direction of its edge, and it cannot move perpendicular to the blade edge. This is implemented by the non-holonomic constraint demonstrated in Figure 3. Here, we see the blade on the \(x\)-\(z\) plane from above, and the dotted line is the traveling direction of blade, \(\vec{d}\).

\[
\dot{x}\sin\theta - \dot{z}\cos\theta = 0, \tag{3}
\]

where \(\theta\) is the angle between \(\vec{d}\) and the \(x\)-axis. \(\dot{x}\) and \(\dot{z}\) are velocity in the \(x\)-axis and \(z\)-axis, respectively, at each point of the blade.
θ direction of the blade depending on the leaning angle of the blade. Therefore, to implement turning, we change the body can turn in the direction tilt. We did not model the exact joining the front and rear points of the blade, there is slight room for turning because of the space between the two edges; therefore, we calculate the impulse to compensate for the deviation of the blade from the line at the next time step. We applied the constraint impulse to the end points of the blade to implement this.

4.2. Relaxation of Non-holonomic Constraints

The shape of the skate blade is very complicated. A skate blade consists of two edges, and there is a hollow area between them. This structure allows for easy turning. There are three contact states according to the ankle orientation: flat, inside edge, and outside edge, illustrated in Figure 4.

Although the skate can only slide along the direction made by joining the front and rear points of the blade, there is slight room for turning because of the space between the two edges; therefore, the body can turn in the direction tilt. We did not model the exact design of the blade. Therefore, to implement turning, we change the direction of the blade depending on the leaning angle of the blade.

When skating, skaters use either the inside edge or outside edge to turn. θ₁ is the inclined angle of the blade on the sagittal plane (x-axis), and θ₂ is the rotation angle on the vertical axis for one simulation time step, as shown in Figure 5. To turn, the skaters must lean their body in the desired direction. At this moment, centrifugal force is generated. We can obtain the θ₁ by analyzing the gravitational and centrifugal forces being applied to the blade.

To meet the non-holonomic constraints described above, the direction of the blade in the current frame and that in the next frame should be the same in the simulation. First, we find the line the current blade generates, and then we calculate the impulse to compensate for the deviation of the blade from the line at the next time step. We applied the constraint impulse to the end points of the blade to implement this.

\[
\tan \theta_x = \frac{v^2}{rg},
\]

where \( v \) is the velocity of the blade, \( r \) is the turning radius, and \( g = 9.8 \text{m/s}^2 \) is gravitational acceleration. Because \( r \theta_c \approx v \Delta \theta \) where \( \Delta \theta \) is a time step, by substituting \( r \) in Equation 4, we can obtain the relationship between \( \theta_x \) and \( \theta_c \).

\[
\tan \theta_x \approx \frac{v}{g \Delta \theta} \theta_c.
\]

Because the tangent function can lead to numerical instability, we assume that \( \theta_x \) is proportional to \( \theta_c \), and use this property to our simulation for turning. For implementation detail, we first rotate the line of the blade by \( \theta_c \), and we enforce that the blade in next time step should lie on the rotated line.

5. Data Acquisition

Tracking reference motion data is a common way to simulate human locomotion and other activities. There are two methods for creating a reference motion to mimic: one is to capture real human motion data, and the other is to make plausible poses manually from user or automatically using optimization with high-level objectives. The method of using motion capture data has benefits when simulating human motion because it provides naturalness and contains principles of human movement. However, to obtain such motion capture data, we must set up a motion capture system. Furthermore, marker occlusion issues can occur during dynamic actions. Therefore, capturing the outdoor sports in a big scene is almost impossible.

Hu et al. [HLL*19] proposed a motion planning method for skiing by adopting real-world skiing skills, such as inclination and angulation, instead of using motion capture data. Skiing is an outdoor sports, so a motion capture system cannot be applied to it. Likewise, figure skating motion is highly dynamic and fast and covers a vast area. Therefore, we can neither apply a motion capture system ourselves nor use any existing motion capture data.

Recently, Peng et al. [PKM*18] proposed a method for obtaining a reference motion from video clips. Unfortunately, because figure skating is composed of dynamic movements, the camera viewpoint changes a lot in most figure skating videos. This makes it difficult to extract the global position and orientation of the human in a video, so we could not directly apply their framework. As a solution to this, we obtain 3D key poses from video, and then generate global translations of each pose using trajectory optimization.

To obtain key poses, we used a method proposed by Kanazawa et al. [KBJM18] called HMR. They used an image as input to the system to find the pose and shape of a human. When the input image is given, HMR outputs 3D human meshes as well as other information including 3D joint positions. We used 3D joint positions and solved inverse kinematics to obtain the joint angles. We found that HMR sometimes generates inaccurate poses, especially in dynamic scenes with high occlusion of body parts. Therefore, after we obtain key poses through the HMR process, we do additional pose editing of critical pose differences manually.
The resulting poses can be represented as $S_q = \{q_1, q_2, \ldots, q_T\}$, where $T$ is the motion length.

6. Trajectory Optimization and Control

6.1. Trajectory Optimization

So far, we only have key poses from videos using HMR. However, these key poses are obtained from different camera viewpoints, so they have no global position information. Also, they are sampled sparsely so the connection between poses is also not smooth.

Therefore it is undesirable to just track each pose in order for a set amount of time. To make a successful simulated motion, we choose several high-level objectives, which are pre-defined window by window. CMA-ES optimizes two consecutive windows at a time and uses only the result of the first window. Then, a fixed duration of 0.5 s. CMA-ES optimizes two consecutive windows.

The objective terms for trajectory optimization are used during a window or only at the end of the window according to skill, and the details are as follows. We set y-axis as the normal direction of the ground, and the opposite direction of gravity.

Each figure skating skill consists of 2 to 5 windows depending on the level of technical skill required, and the duration of each window is given by the user. The range of the given duration is between 0.2 and 0.5 s for window, which is different to previous work, using a fixed duration of 0.5 s. CMA-ES optimizes two consecutive windows at a time and uses only the result of the first window. Then, the optimization begins for the next two windows using the result of the first window and so on. The optimization process ends once the last window is optimized.

To simulate each figure skating skill successfully, we choose several high-level objectives, which are pre-defined window by window. Next, we minimize the weighted sum of chosen objectives.

$$E = \sum_k w_k E_k,$$

where $k$ is a number of chosen objectives for a window.

Here, we describe the objective terms for trajectory optimization beforehand. The objectives are used during a window or only at the end of the window according to skill, and the details are as follows. We set y-axis as the normal direction of the ground, and the opposite direction of gravity.

**End Effector On Ground.** $E_{\text{BladeOnGround}}$ enforces a specified blade to be in contact with the ground.

$$E_{\text{BladeOnGround}} = \sum_{j \in \{\text{rb, lb}\}} |p_{jY} - \text{proj}_{\text{rb, lb}}(p_j)|^2,$$  \hspace{1cm} (7)

where $p_{\text{rbY}}$ and $p_{\text{lbY}}$ are the heights of the right and left blades, respectively.

**COM Height.** We used the height of the center of mass (COM) as a regularizer.

$$E_{\text{COMH}} = |cY - \bar{c}Y|^2,$$  \hspace{1cm} (8)

where $cY$ is the simulated COM height, and $\bar{c}Y$ is the desired COM height.

**Maximize COM Height.** For a higher jump, we maximize the COM height using $E_{\text{MaxComH}}$.

$$E_{\text{MaxComH}} = -|cY|^2.$$  \hspace{1cm} (9)

**Body On COM.** $E_{\text{OnCOM}}$ is a term that makes a body part as close as possible to the COM.

$$E_{\text{OnCOM}} = \sum_{j \in \{\text{head, rb, lb}\}} |\text{proj}(c) - \text{proj}(p_j)|^2,$$  \hspace{1cm} (10)

where $\text{proj}(p)$ is the projection of a point $p$ onto the xz-plane, and $p_{\text{head}}$, $p_{\text{rb}}$, and $p_{\text{lb}}$ are the positions of the head, right blade, and left blade, respectively.

**Maximize Angular Momentum.** For jump motion, we use $E_{\text{MaxAM}}$ which maximizes angular momentum.

$$E_{\text{MaxAM}} = -|L_Y|^2,$$  \hspace{1cm} (11)

where $L_Y$ is the angular momentum of the character with respect to the COM.

**Maximize COM Velocity.** $E_{\text{MaxVel}}$ maximizes the COM velocity to get the maximum power to the normal axis for jumping.

$$E_{\text{MaxVel}} = -\text{sgn}(v_Y)|v_Y|^2,$$  \hspace{1cm} (12)

where $v_Y$ is the velocity of the COM, and $\text{sgn}(v_Y)$ is the sign of $v_Y$.

**COM velocity.** We can penalize COM velocity difference with desired COM velocity using the $E_{\text{ComVel}}$ term.

$$E_{\text{ComVel}} = |v - \bar{v}|^2,$$  \hspace{1cm} (13)

where $v$ is the simulated COM velocity and $\bar{v}$ is the desired COM velocity.

**Heading direction.** We can give the heading direction of a character.

$$E_{\text{PelvisHeading}} = |\alpha - \bar{\alpha}|^2,$$  \hspace{1cm} (14)

where $\alpha$ and $\bar{\alpha}$ are the simulated and desired pelvis heading direction. We define the heading direction of the pelvis as the x-axis in the local coordinates of the pelvis.

**Track Pose.** To regularize the solution of optimization, we minimize the differences between the key poses $q_{\text{key}}$ and the sampled poses $q$ during the optimization process. Note that when optimizing every window, we use the key pose tracking term $E_{\text{TrackPose}}$ in common.

$$E_{\text{TrackPose}} = |q - q_{\text{key}}|^2.$$  \hspace{1cm} (15)

© 2019 The Author(s)
Computer Graphics Forum © 2019 The Eurographics Association and John Wiley & Sons Ltd.
6.1. Forward stroking

Forward stroking is a skating motion that alternates propelling and gliding to move forward. After extracting a propelling pose and a gliding pose when left foot stance, we get poses when right foot stance by inverting the left foot stance poses. Forward stroking has four windows: right-propelling, right-gliding, left-propelling, and left-gliding. In propelling window, we use $E_{BladeOnGround}$ for both blades and $E_{OnCom}$ for the left blade. Then, the right blade pushes the ground to gain momentum and the skater model glides using the left blade using the momentum in the second window, where $E_{BladeOnGround}$ and $E_{OnCom}$ are used for the left blade. In the third window, the skater model crosses the right leg over the left. In this step, we use the same objectives used in the first window, except $E_{OnCom}$ is used for the right foot instead of the left one. The character repeats these steps iteratively for turning.

6.1.2. Crossover

Crossover is a basic skating skill used for turning. It consists of three windows. It is conducted as follows. In the first window, both blades are in contact with the ground. We use $E_{BladeOnGround}$ and $E_{OnCom}$ for the left blade. Then, the right blade pushes the ground to gain momentum and the skater model glides using the left blade using the momentum in the second window, where $E_{BladeOnGround}$ and $E_{OnCom}$ are used for the left blade. The final window is the end of the jump, landing, and balancing. The same objectives as in the fourth window are used during the final window.

6.1.3. Three-turn

Three-turn is a skill used to turn from forwards to backwards, or vice versa. We assume that an initial velocity is given. For a three-turn, we use two windows. The orientation of the torso is different at each window, and because of this, the character can gain angular momentum. We use $E_{BladeOnGround}$ and $E_{OnCom}$ for the skating leg which is left blade in this case. To skate backwards while changing the heading direction, $E_{ComVel}$ and $E_{PelvisHeading}$ are used. For the first window, we use $\bar{\alpha} = \frac{1}{\sqrt{2}} (1, 0, 1)$ to calculate $E_{PelvisHeading}$, and $\bar{\alpha} = \frac{1}{\sqrt{2}} (-1, 0, -1)$ for the second one. $E_{ComVel}$ is the same for both windows.

6.1.4. Jump

Jump is a complicated figure skating skill, so it is composed of five windows. The first window is the take-off step, which prepares the jump. To jump higher, the height of the COM must be increased significantly. At the first window, we set $c_y = 0.64$ m in $E_{ComH}$, and $E_{BladeOnGround}$ is used for the left foot for balancing before the jump. At the end of this window, the character must jump upwards and rotate simultaneously for a high jump and fast rotation. To accomplish this, we use $E_{MaxVel}$ and $E_{MaxAM}$ at the end of the window.

The second, third, and fourth windows are in-air stages. The second window is the very start of the jump in the air and ends at the maximum height, $E_{MaxAM}$, $E_{MaxComH}$, and $E_{OnCom}$ for the head and both blades are used for this window. The third window is the middle part in the air, where the same objectives as in the previous window are used except for $E_{MaxComH}$. In the fourth window, the skater model prepares for landing with the right leg, so $E_{BladeOnGround}$ and $E_{OnCom}$ are used as objectives for successful landing at the end of the window. The final window is the end of the jump, landing, and balancing. The same objectives as in the fourth window are used during the final window.

6.2. Control

As a result of trajectory optimization, we can create physically simulated motions for each figure skating skill. However, these results are not robust, so a small perturbation can lead to simulation failure. Although Wang et al. [WPH10] showed that the presence of noise in the optimization process could make a controller more robust, noise also makes the trajectory optimization fail in our cases because of the existence of the non-holonomic constraint.

To make robust controllers for each skill, we apply a DRL approach. Reinforcement learning determines the action policy that maximizes the discounted sum of future rewards from given agent and environment states. Recently, Peng et al. [PALvdP18] proposed a DRL-based control framework, which can imitate the diverse human motion capture data, including cartwheels and acrobatics. We have dynamic motions as well like jumps; therefore, we used this framework to find our controllers. State $s = (\phi, p, v, R, \omega)$ in our system consists of a phase variable $\phi$ that indicates a progress of the reference motion, accumulated vectors of the body link positions $p$, velocities $v$, orientations $R$, and angular velocities $\omega$, which are represented in the local coordinate of the root body (a pelvis in our case). The output of the neural network is an action vector $a$ which is a displacement of a pose vector $q$. Then the character tracks $q_{des} = q + a$ by PD control. Because we follow reward terms used in previous work, please refer to the original paper [PALvdP18] for details.

To control each skill that is robust to various speeds, the simulated character mimics the reference motion in which the average speed is randomly changed during training the network.

7. Experiments and results

Our skater model has 57 degrees of freedom (DOFs), including 6 DOF for a floating root body and 3 DOF for each joint. The skater model is 1.5 m tall and weighs 50.07 kg. We put a 5.4 cm tall, 20.8 cm long blade under each foot to model the skates. The weight of blade is 1 kg in our simulation (Figure 6).
To create the ice environment, we set the friction coefficient of the ground to 0.02 [Mil08]. We used the Dart [LGH*18] open source engine for the simulation. Simulation time rate is 1.2 kHz. For PD gain, we use (600, 49), except for the jump skill, which requires more force to reach a sufficient height for rotation. We used (1000, 60) for PD gain in the jump skill. We run the simulation on a PC with an Intel Xeon CPU E5-2687W v4 (3.0GHz). The population size of CMA-ES optimization is 12, and the optimization is run in parallel with 24 processes. The optimization takes two to six hours for each figure skating skill. To train the neural network of DRL, we use proximal policy optimization (PPO) [SWD*17] algorithm implemented with PyTorch [PGC*17]. We use three fully connected layers and 128 units in each layer for the neural network. Training the neural network usually takes one day, but not exceeds two days for plausible results.

**Forward stroking.** Forward stroking is a simple skating motion to move forward (Figure 7a). It consists of two main steps; propelling and gliding. When propelling, a skater can get a momentum to go forward by pushing the ice with a skate blade. Using this power, he/she can glide with one leg and move forward by repeating propelling and gliding. To produce longer stroking motion, we first optimized one cycle of stroking using CMA-ES, and then let DRL mimic the optimized result repeatedly.

**Crossover.** Figure skating movements mostly consist of rotations. The trajectories of the blades are curves. Crossover is the basic skill that a skater uses to move along the curve (Figure 7b, 7c). First, they hold the left foot on an axis and roll the right foot while bending the left knee. At this time, the skater gains momentum to go forward. While sliding, they lift the right foot up and cross it over so that the two legs become reversed. Then, they place the left foot on the starting position and repeat these steps. Note that the upper body looks towards the center of the circle to rotate smoothly. We optimized two cycles of crossover using CMA-ES, and then made long crossover motion by letting the character track the optimized result repeatedly using DRL.

**Three-turn.** The three-turn is an easy way to change the heading direction of a skater’s body from forwards to backwards or vice versa (Figure 7d, 7e). Before performing jump, the skater turns backwards to prepare (except with the Axel jump). At this moment, the skater uses the forward three-turn. After completing jump, the
skater lands in backward pose and he or she can change the heading direction to forward using the backward three-turn.

**Jump.** The jump skill is scored the highest in a figure skating sequence, and it is the most elaborate skill but also the most difficult. In figure skating, there are six jump techniques and two initial jump methods (edge jump or toe jump). Each technique has a different level of difficulty. The difficulties of the techniques are as follows: axel > Lutz > flip > loop > Salchow > toe loop. Moreover, depending on the number of rotations, there are single, double, and triple jumps. We try to simulate a double Salchow jump, which is an edge jump (Figure 7f). The Salchow jump involves jumping to the inside edge of the left skate in the backward state and landing on the right foot.

![Jump in place.](image1.png) ![Jump at 0.5 m/s.](image2.png) ![Jump at 1.0 m/s.](image3.png) ![Jump at 1.5 m/s.](image4.png)

**Figure 8:** Robustness for various speeds.

**Robustness.** Using DRL, we found robust controllers for each figure skating skill. First, we generate the controller with robustness to speed. When generating each episode while training, we train the episode by randomly changing the traveling direction speed of the motion that is obtained from the trajectory optimization. Note that we used one jump motion to learn robust controllers for diverse speeds. As a result, we can control the three-turn skill from 0 to 4.5 m/s and the jump skill from 0 to 2.0 m/s (Figure 8).

To demonstrate robustness against external perturbation, we push the character several times while it conducts a crossover. Each external force is 400 N and is applied at the torso from the side for 0.1 seconds. Our controller is robust enough to survive after three impulses are applied (Figure 9). We conducted same experiment for the jump. We apply an force at the pelvis for 0.2 seconds while the character is in the air mid-jump. The character successfully jumped with a 100 N and 150 N force but failed with a 200 N force.

![Figure 9: Robustness to external perturbation while simulating crossover.](image5.png)

**8. Discussion**

In this paper, we proposed a framework for figure skating simulation that uses videos as input. To simulate a skater character on an ice surface, we constrain the movements of the blade using non-holonomic constraints. We were able to reproduce figure skating motions with only a few key poses obtained from videos. Our framework used pose estimation to achieve naturalness and used high-level objectives for each motion for trajectory optimization. Using DRL, we obtained robust controllers for each figure skating skill. To the best of our knowledge, this is the first physics simulation of figure skating using a full body human-like character on an ice surface. We demonstrated our system by showing diverse figure skating skills, including jump.

Nevertheless, there are several limitations. Currently, still images must be selected manually from videos. From these images, 3D joint positions for key poses are extracted using HMR. However, it seems that the camera viewpoint is a critical factor in pose estimation from a 2D image. For example, when the camera films a skater from the side, arm posture is not accurately estimated. Furthermore, HMR did not work well for estimating acrobatic poses, such as spiral. Kanazawa et al. [KBM18] used several in-the-wild image datasets for training, but to enhance the performance, it is better to add many acrobatic poses from figure skating or ballet. In the data acquisition process, 3D pose estimation from a single 2D image is not accurate, so manual pose editing is inevitable. Therefore, effort to automate this process is required.

Hu et al. [HLL+19] applied a ski-snow contact model to their skiing simulation. It is known that the blade melts the ice, turning the ice into water quickly so that the skater can slide. However, the exact mechanism of ice slipperiness has not been revealed clearly [Mil08]. Therefore, we could not apply real ice-blade physics, and we instead set a low friction coefficient.

We didn’t handle self-collision issues in this work. If we resolve this in the near future, motion quality will be much better, especially crossover.

Although we successfully simulated the dynamic figure skills of jump, we failed to generate spin because of the friction issues. The spin is a major element in figure skating along with jump, and there are many kinds of spins and jumps. To simulate those kinds of spins and jumps, we need more precise modeling and delicate edge control method, which is good subject for the future work. For simplicity, we designed the skate blade as a rectangular shape. To make a simulated character be able to turn under the non-holonomic constraints, we allow some exceptions using the constraint modification introduced in section 4.2. However, this can cause artificial energy to the system. To mimic figure skating motions precisely, we believe that blade design is a crucial factor. A real figure skating blade has a curved shape and a toe pick on the front, which plays an important role in skating, especially in spins or jumps. Moreover, a figure skating blade has two edges. Therefore, for fine control of the skate blade to represent accurate jumps or spins, the in-edge and out-edge should be distinguishable.
In our framework, trajectory optimization and DRL methods are used separately. However, there have been several studies that combine trajectory optimization and policy search methods [TGLT14, MT14, WPKL17]. In particular, Won et al. [WPKL17] reported that using DQL with an evolutionary strategy gives fast convergence and finds a better policy than when using DQL only. In future work, we would like to adopt these approaches.

We made a robust controller for each figure skating skill using our framework. Based on that, we will try to build a short program, a figure skating event which consists of jumps, spins, the combinations of each skills, and step sequences for 2 minutes and 50 seconds. Also, controlling a figure skater interactively will be an interesting work.

Acknowledgements

This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the SW Starlab support program (IITP-2017-0-00878) supervised by the IITP (Institute for Information & communications Technology Promotion).

References

[CMJMS84]ole L., Meixel Jr G. D., Morris T. L., Stoner L. J.: Figure skating jump optimization through computer simulation. In ISBS-Conference Proceedings Archive (1984), vol. 1. 2

Yu et al. / Figure Skating Simulation from Video


