IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 12 January 2023, accepted 26 January 2023, date of publication 30 January 2023, date of current version 2 February 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3240589

RESEARCH ARTICLE

Interactive Character Path-Following Using Long-Horizon Motion Matching With Revised Future Queries

JEONGMIN LEE[1,2], TAESOO KWON[2], AND YOONSANG LEE[2]

[1]Samsung Research, Seoul 06620, South Korea
[2]Department of Computer Science, Hanyang University, Seoul 04763, South Korea

Corresponding authors: Yoonsang Lee (yoonsanglee@hanyang.ac.kr) and Taesoo Kwon (taesoo@hanyang.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) Grant through the Korea Government Ministry of Science and ICT (MSIT) under Grant NRF-2019R1C1C1006778 and Grant NRF-2020R1A2C1012847.

ABSTRACT We propose a method for interactively generating a character motion that follows a user-specified path based on motion matching. Unlike basic motion matching that finds the best frame considering only the current state and current control input, our long-horizon motion matching algorithm is performed recursively over multiple levels and finds the matching path that minimizes the total distance, to find the best frame for a longer future interval. The matching query is carefully revised from the raw query directly extracted from the user path to address the following aspects: inherent differences between an actual moving path of a human and a user-drawn path, and possible user errors such as a path drawn with excessive speed or with abrupt changes in direction. Also, two interactive control modes for path-following characters are proposed. The combination of our long-horizon motion matching and carefully revised queries enables smooth and reliable interactive path tracking for a variety of paths. We show the effectiveness of our method using various experiments such as path-following with different shaped paths, game-like interactive control demos, and ablation studies.

INDEX TERMS Character animation, interactive path-following, motion matching.

I. INTRODUCTION

Drawing a path for a target to follow is one of the interfaces that a user can use to control a character or object in an interactive application such as a game. A method commonly used to control a game character is an *instantaneous control* method in which the moving direction and speed are inputted with a gamepad or mouse and keyboard at every moment. Contrary to this, in this *path-following* method, the user can input the movement of a target for a certain duration in the future at once through a pointing device or a touch screen. Using this, as seen in the famous classic iPad games Harbor Master [1] and Flight Control [2], simultaneous interactive control of multiple objects in complex scenarios such as controlling multiple moving objects to pass through the same area with a small time difference is also possible.

Although approaches based on neural networks trained with motion capture data are receiving a lot of attention for generating the full-body motion of an instantaneously controlled character [3], [4], a much simpler approach called *motion matching* [5] is currently popular in gaming industry due to its simplicity and relatively high motion quality. This method periodically searches the best next motion frame considering only the current character state and current control input. Since motion matching is essentially a method to find the best option available at the present moment, it is ideal for an instantaneously controlled character that receives the movement direction and speed at every instant.

However, in the case of the path-following problem, there are many difficulties in directly applying the standard motion matching technique as it is. First, the character needs to

The associate editor coordinating the review of this manuscript and approving it for publication was Shihong Ding.

switch motions naturally while continuously following the path specified by the user. However, motion matching uses a greedy search to select the current best option, which is highly unlikely to be the best option in the long run, when the future section of the user-drawn path is also considered.[1] Second, depending on the user's immersion in the interactive application and how quickly a task needs to be completed, paths may be drawn by the user with excessive speed or abrupt changes in direction. A raw motion matching query directly extracted from such a user-drawn path would produce a motion in which the character does not follow the path properly.

In this work, we propose a method for interactive character path-following based on motion matching. Specifically, we propose a long-horizon motion matching (LHMM) algorithm to find the best next frame considering the current character state as well as the predicted state in the longer future. We also propose a method to revise the future query to create a stable motion even when an extreme path is entered.

The user-specified path is drawn on the ground plane using a pointing device such as a mouse or stylus pen. Our method generates motion of the character moving along this path input with various locomotion styles, such as walking, jogging, or running, depending on the movement speed embedded in the path.

Unlike basic motion matching that finds the best frame considering only the current state and current control input, our LHMM algorithm is performed recursively over multiple levels and finds the matching path that minimizes the total distance, to find the best frame for a longer future interval. Specifically, for each candidate frame found in the previous matching step, the next k candidate frames are recursively searched until l steps are reached for each search path. Among a total of k^l matching paths searched, the matching path with the smallest cumulative cost is temporally followed by selecting the first frame of the path as the next transition frame. As a result, it is possible to find a semi-optimal solution for the future time interval corresponding to l matching steps, which enables smooth and stable path-following.

The matching query is carefully revised from the raw query directly extracted from the user path to address the following problems. First, the raw time-stamped path can generate motion with unnatural velocity changes near the corners because those changes in hand-drawn paths tend to be more abrupt than the movement of a person walking on a similar path. We propose to preprocess input paths to mitigate these artifacts. Second, the speed of drawing a path by hand on the rendered ground is usually much faster than the speed of human movement. Even when properly scaled down, there can always be portions of a path drawn at a speed that exceeds the maximum possible speed of a person's movement. We carefully adjust the timings and the extent of future positions in the query to avoid excessive progression of path following, resulting in poor quality subsequent matches. Finally, if there is a sharp corner on the user-drawn path, the character is often unable to follow the path well around the corner. We propose to modify the future positions of the query so that the character approaches the corner vertex more closely.

For this "interactive character path-following" task, we propose two control modes: *Local Control* mode, which allows egocentric control over the character given its current state, and *Global Control* mode, which is useful for controlling a global position of the character. We also provide three different ways to specify the character's facing direction on the input path: typical tangential direction, user-specified direction with a joystick, and *DirectionNet* which infers the natural facing direction of the character on the input path.

There have been some character animation studies showing path-following demos [6], [7], [8], [9], [10], [11]. However, to the best of our knowledge, none of the studies have focused on the path following task itself and demonstrated different types of path-following examples, including excessive path input. Focusing on path-following, our method enables a diverse set of interactive character path-following demos with extreme paths where the speed changes abruptly or a path contains sharp corners.

Our contributions can be summarized as follows:

- A long-horizon motion matching scheme that finds the best frame for a longer interval rather than at the moment.
- The future query revision process and preprocessing of a user-drawn path for reliable character path-following for various paths including extreme cases.
- More diverse character path-following results than previous studies, including multi-character control, game-like interactive control, and path-following with different motion styles.
- Two new control schemes for interactive character path-following: local control and global control.

II. RELATED WORKS

One of the popular approaches to synthesize natural human movements is to utilize motion capture data. Because a motion capture system records the movement of a human subject as it is, the captured motion is inherently very natural and realistic. Therefore, researchers have proposed synthesizing realistic character animation using motion capture data in various ways including constructing a graph structure from a large collection of motion capture data and searching and blending motions through the graph [12], [13], training regression models with motion capture datasets [7], [14], [15], and using motion capture data to guide the movement of a physically simulated character [16], [17].

This section reviews motion synthesis studies that use motion capture data, from motion graphs to state-of-the-art deep neural network-based approaches, as well as studies

[1] For example, let's assume that there are two candidate frames A and B, where A is the start frame of a regular and steady step, and B is the start frame of a quick and short step. Even if it is frame A that better satisfies the current requirements of the user-drawn path, frame B may be the optimal solution in the long-term when the direction of the user-drawn path changes rapidly within a certain period in the future.

utilizing several user input types for motion synthesis, specifically path-following.

A. MOTION GRAPHS AND ITS VARIANTS

Motion graphs calculate all possible transitions between the motion frames in a large motion database to build a graph structure so that every graph traversal path can produce a plausible motion stream [12], [13], [18]. Because the process of near-optimal search in a large-scale graph can take a long time, many algorithms such as A* algorithm [19], dynamic programming [20], [21], state-space search [22], [23], or min-max search [24] have been experimented with to accelerate the search. Pre-computational methods are also helpful when user-inputs are relatively low-dimensional [25], [26], [27].

Motion matching [5] is another variant of the motion graph. Unlike most other motion graph algorithms which perform multi-depth searches to optimize long-term objectives, motion matching uses a greedy search to make transition anywhere at any time with only immediate rewards in mind. Under this simplification, the entire database can be searched efficiently for a fast response, unlike motion graphs that only use heavily pruned transitions and thus can lead to a delayed transition. Due to its fast transition and simple implementation, motion matching is one of the popular methods for a game development.

However, a greedy search can obviously make suboptimal decisions, and such a possibility should be mitigated using a lot of intuition and heuristics. As such, a feature vector plays an important role in motion matching, which is a task-specific representation of a motion frame. It is often dependent on motion datasets, and is manually designed for best performance. For example, a locomotion dataset works well with a low-dimensional feature space having only the feet and root information of the character. Different features can be used for other motion styles such as soccer games [28], parkour maneuvers [29], close interactions between characters [30], or even quadruped locomotions [4]. In this paper, we propose a long-horizon motion matching scheme to overcome the drawback of greedy search in standard motion matching. We use a low-dimensional feature space similar to that for locomotion datasets but extract motion matching queries from a user-drawn path to provide more flexibility and controllability.

B. LEARNING-BASED APPROACHES

Discrete samples in a database can be difficult to use for a continuous and precise control. Many researchers find that it can be useful to model behaviors in a continuous manifold space. Linear methods such as Principal Component Analysis (PCA) [31], [32] compose full-body controller controlled by low-dimensional signals. Kernel based methods [33], [34] have synthesize motions in various non-linear contexts. Gaussian process is a notable non-parametric model that synthesize motions from low-dimensional latent variables that accounts for uncertainty in the model [14], [33].

Recently, deep learning-based methods are popular in data-driven animation. Neural networks can learn the manifold of a large motion database to directly generate a sequential motion or to represent the database in an efficient way. An auto-regressive model takes the current posture as an input to generate the next posture [8], [11], [35], [36]. However, auto-regressive models tend to show 'dying-out' effect [35], where the output motion gets blurry in the long sequence of motion. Phase network [7] or gating network [3], [15] alleviated this by mixing multiple network outputs using time-varying weights. Starke and his colleagues further extended a gating network to interact with diverse environments or another character [37]. Ling and colleagues used deep reinforcement learning with a latent space learned using a variational auto-encoder [9]. Deep reinforcement learning is also widely used with physically simulated characters [17], [38], [39], [40]. Unlike these deep learning-based methods, our method does not directly create a posture through a neural network. Instead, we propose to use *DirectionNet*, an RNN-based network, to infer the facing direction of the character, for natural transition of facing direction.

Deep learning is also often joined with graph-based approaches. Since motion graph or motion matching can find diverse paths from a specific motion database, the output motions can be used as a dataset to train generative models [4], [36], [41], [42], [43]. Holden and his colleagues combined a motion matching algorithm and a learned database without explicitly storing motions and feature databases [4].

C. USER INPUT TYPES AND PATH-FOLLOWING

Various types of user input have been used for synthesizing character locomotion. One frequently used input type is to give an instantaneous control signal to the target character at every moment, such as a speed and direction input from a gamepad or joystick [3], [7], [15], or a full-body posture input from a RGB or depth camera [44], [45]. Another commonly used input type is to specify where the target character must reach in the future. This tasks has recently been accomplished using deep reinforcement learning in animation studies [9], [43].

We want to focus on the third type of input, path following, which has the characteristics of both of the previous two methods, like the former, which can specify the movement at every moment, while at the same time, control all the movements up to a certain point in the future, like the latter. The instantaneous velocity control provides some leeway in terms of goal achievement, in that the character can freely drift from its original position and achieve the target direction with a slight delay, but it allows the user to control the character only momentarily. Specifying the path for the character can be somewhat difficult to get the character in the right position at the right time, but it can provide the user with wider controllability. Although some studies show path-following demos, the input path often is not time-parameterized, and thus the character can freely change its moving speed. A few recent

studies [6], [7], [8], [9], [10], [11] include similar tasks to ours but how they composed the motion via trajectory is somewhat vague. As long as we understand, the result videos show near constant-speed trajectories, and the character usually faces front. These constraints can limit the use of database on specific clips. Chen and his colleagues demonstrate a character dancing through a trajectory [42]. The character performs impressive motions by searching motion graphs, although the speed of the character is not important in this example. The learned motion matching [4] also includes a trajectory tracking demo, but how they composed a query is unclear. In this paper, the temporal sample points in the trajectory are used for calculating instant speeds, and a method to allow the character to face other directions are propose, away from the tangential direction of the given path. By doing so, our method better exploits the dataset while the user gets more controllability over the character.

III. OVERVIEW

The path drawn by the user is first encoded as 2D positions sampled at regular time intervals in the preprocessing step, and then goes through a process of reducing artifacts due to the characteristics of the input device (Section V). In runtime, LHMM is executed every N frame to search for the best next motion frame, and then the motion is transitioned to that frame (Section IV). The raw future query extracted from the input path is revised before being used as a query in LHMM (Section VI). Using this method, the user can interactively control the character by drawing paths in two different ways (Section VII). In the rest of the paper, the LHMM algorithm is first described in Section IV, that can be used for general purposes other than the path-following task, and then path-following specific input path preprocessing, future query revision, and interactive control modes are described respectively in Sections V, VI, and VII.

IV. LONG-HORIZON MOTION MATCHING

This section first briefly describes the basic motion matching, and then describes the proposed long-horizon motion matching (LHMM) algorithm.

Basic Motion Matching (BMM). This technique first extracts a simple feature for every frame in the motion database and stores it in the feature database. At runtime, at every N frames, it searches the feature database for the next pose frame with the closest feature to a search query, which is performed in a greedy manner only considering the current moment. The query consists of the current character state and control input, usually collected from an input device such as a gamepad or keyboard [5]. Please refer to Appendix A for the details on the basic motion matching.

Long-Horizon Motion Matching (LHMM). Similar to BMM, LHMM uses motion database and feature database, and executes a matching step every N frames to find the next pose frame that will be immediately followed. However, unlike BMM, LHMM searches for the best next frame considering not only the present moment but also the expected character state and control input for a certain future time period. Instead of simply including control input for a longer future horizon in a motion matching feature and query, multiple candidate frames selectable at multiple future moments are searched in a recursive way by stacking future character states and control inputs.

Algorithm 1 Long-Horizon Motion Matching

Input: *pose*: character pose at the time of querying *ctx*: contextual information such as user input or environment k: number of candidates (nearest neighbors) at each level l: level of LHMM *fdb*: feature database containing features for all motion frames *mdb*: motion database containing poses in all motion frames

Output: *matched_frame*: best frame that minimize the sum of distances on all levels *total_distance*: sum of the distances at that time

```
function LHMM(pose, ctx, k, l, fdb, mdb)
    q ← compute_query(pose, ctx)

    if l = 1 then
        ▷ f: nearest frame, d: distance between q and fdb[f]
        (f, d) ← kNN_search(q, 1, fdb)
        return (f, d)
    else
        min_d ← ∞
        {(f_i, d_i)}_{i=1}^k ← kNN_search(q, k, fdb)
        for each (f, d) ∈ {(f_i, d_i)}_{i=1}^k do
            ▷ get aligned next end pose mdb[f+N] (transformed by pose.root − mdb[f].root, N : motion matching interval)
            n_endpose ← align(f, pose, mdb)
            n_ctx ← update_context(ctx, ...)

            d ← d+LHMM(n_endpose, n_ctx, k, l − 1, fdb, mdb)[1]
            if d < min_d then
                min_d ← d
                matched_frame ← f
            end if
        end for
        return (matched_frame, min_d)
    end if
end function
```

Algorithm 1 describes the LHMM process. The function LHMM() finds the next frame such that all the requirements including the current character pose and future control inputs are best satisfied in the future section where motion matchings of depth l are performed recursively. A query q is generated from each state and contextual information of the moment when the LHMM() function is called, and k candidate frames closest to q are selected with k-nearest neighbor

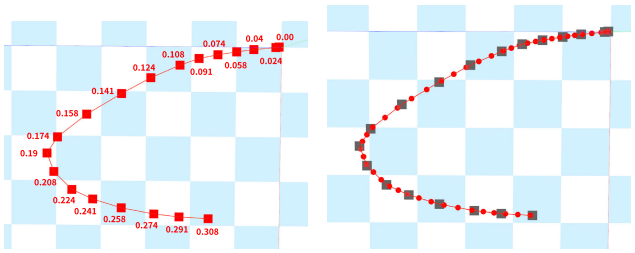


FIGURE 1. Construction of the time-stamped user-drawn path. (a) Raw locations and corresponding times (in seconds) recorded by our application. (b) Path points resampled at 150 Hz (red circles). Gray boxes indicate the raw locations.

(kNN) search. The next steps then start from those candidate frames. This way, the same LHMM() function is called again for the next step where the motion is progressed for the motion matching interval N. This search is semi-optimal because only k-nearest candidate frames are searched for each step rather than using an exhaustive exploration of all frames. *total_distance* returned by a specific LHMM() function call in the $l = i$-th step is the shortest distance among the distances obtained by adding the shortest cumulated distance returned by the LHMM call of $l = (i-1)$-th step to each distance between the query q and a next frame candidate f_i used in the call. Since the $l = 1$-th step is the last step in the recursive call, search time can be saved by finding only the closest frame. As a result, the frame with the smallest total distance among all k^l search paths is selected as the next frame. We'd like to note that LHMM reduces to BMM if $l = 1$.

Note that in this paper, the LHMM algorithm is applied only to the path-following experiments, but this algorithm can be used not only for path-following, but also for any motion matching problem where the control input can be computed at future moments by implementing compute_query() and update_context() for the problem. Please refer to Appendix B for the implementation of these two functions for our path-following tasks.

V. PREPROCESSING USER-DRAWN PATH

When the user draws a path, the position and time of every input moment are recorded so that the moving speed of the character can be controlled by time-varying queries. This raw time-stamped path is transformed into a series of regularly sampled way-points for query construction and visual feedback on speed. Additional preprocessing is necessary to compensate for the differences in the characteristics of a user-drawn path and an actual trajectory of a moving character.

A. CONSTRUCTING TIME-STAMPED USER-DRAWN PATH

Even if the user draws a continuous path on the rendered ground using a pointing device, the computer records the cursor position at discrete timings. One practical problem here is that, in most UI frameworks, the cursor movements are not recorded at regular time intervals (i.e., the mouse move events are not sent at regular time intervals). We record the time at which the cursor event is reported together with the 3D position on the ground obtained from the cursor position. When the trajectory input is completed, the positions named *path points* are resampled at a fixed frame rate (=150 Hz) on the entire path using linear interpolations between two adjacent input points. The user can also visually check the input speed from the path points displayed on the ground (Figure 1).

Since the character is drawn relatively small in the rendering window, if the user draws a path at the speed at which the character actually moves within the window, the user has to draw the path too slowly, causing inconvenience and discomfort. Therefore, it is useful to allow the user to specify the speed of the character using a constant scaling factor. The degree of scaling can be set differently depending on the screen size, the tendency of each individual user, and the type of motion to be generated. In our experiments we used 0.2 as an example of a scale factor (i.e., using the path points sampled at 150 Hz as if they are sampled at 30 Hz). Changing the scaling factor only changes the adequate drawing speed the user feels, and all subsequent path preprocessing and query construction processes work the same.

For convenience, in the remainder of this manuscript, we will use frames instead of seconds as the unit of time (i.e., the first path point at 0 seconds on the input path corresponds to frame 0, the second path point at $1/30$ seconds corresponds to frame 1, the third path point at $2/30$ seconds corresponds to frame 2, and so on). The user-drawn path is expressed as a function $\mathbf{p}(i)$ of the frame i, meaning that the path point corresponding to frame a is denoted by $\mathbf{p}(a)$.

Even on some sections of this five times slower path, however, the speed on the path may still exceed the maximum speed an actual person can move. Handling of such a case is described in Section VI-A and VI-B.

B. SMOOTHING USER-DRAWN PATH

There is a style difference between a user-drawn path and an actual moving path of a person, in terms of speed change. For example, an input device such as a mouse is generally light in weight and there is friction between the mouse and the mouse pad, so the speed change is abrupt near sharp corners. On the other hand, humans are much heavier, and the whole body is underactuated accelerating only by the contact forces between the feet and the ground. So the speed change around the corner is much more gentle (Figure 2). Therefore, if motion matching is performed using a query extracted simply from the raw time-stamped user drawn path, a jerky motion with sudden speed changes may be generated.

In order to make the user-drawn path more smooth as in the actual human movement, a Gaussian filter (sigma = 3, filter width = 19) is applied to the path points obtained in Section V-A (Figure 2). When applying the Gaussian filter close to the boundary of the path, the first two and the last

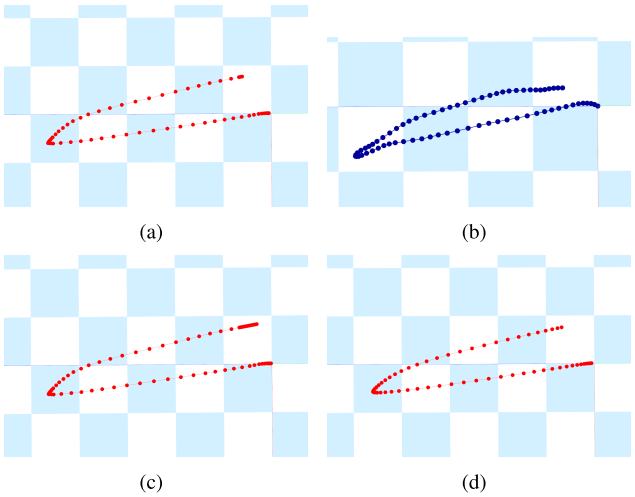


FIGURE 2. Smoothing the user-drawn path. In this figure and all the following figures, the red dots on the path represents the path points. (a) A raw time-stamped user drawn path. (b) The root path of the character moving along a similar path for comparison. (c) The extended user-drawn path before applying a Gaussian filter. (d) The final smoothed user-drawn path. Notice the difference from (a) in the speed change near the corner.

two path points are extrapolated, in order to use the velocities at both ends and also to prevent the length of the user-drawn path from being shortened due to filtering. We compare the results with and without smoothing in Section VIII-F.

VI. FUTURE QUERY REVISION FOR PATH-FOLLOWING

Features for Path-Following. Motion matching features can be designed in different ways depending on the type of motion the database contains, and this process is often relies on human intuition [5], [28], [29], [30]. For our interactive path-following task, the feature design for locomotion proposed in [5] is used, which encodes a full-body posture in each frame into a 27-dimensional feature vector containing the position and velocity of the feet, the velocity of the root, and the position and direction of the root at three different future frames. Our motion database consists of motion clips resampled at 30 Hz, and $i+10$, $i+20$, and $i+30$ are used as the three future frames at frame i when constructing the feature database. Please refer to Appendix C for the details of matching features. Although most of our experiments use this form of features, this is not an essential part of our future query revision process. As long as a feature includes multiple future positions and directions, it can be used in this process.

In our system, the future part of the query (related to the three future frames) is extracted from the user-drawn path and revised by our revision process. The remaining current part of the query comes from the current character state.

Raw Future Query. At runtime, at the current frame i (assuming that the animation starts at frame 0), one naive way to extract the future part of a query from the user-drawn path is to directly use the path points at $i+10$, $i+20$, and $i+30$ for calculating the three future positions and facing directions. However, this raw future query leads to a low-quality motion that does not follow the path well due to various reasons.

In this section, we propose methods to tackle this issue. Sections VI-A, VI-B, and VI-C describe how to revise the future positions of the query. Section VI-D describes how to compute the future forward-facing directions.

A. COMPUTING DESIRED POSITION OF CURRENT CHARACTER

To extract the future part of a query from an input path, first, it is necessary to find the desired position $\mathbf{p}_d$, which is the path point corresponding to the current position of the character, and its frame i_d (i.e. $\mathbf{p}_d = \mathbf{p}(i_d)$). This way, the future query can be obtained from the path points that follow $\mathbf{p}_d$. Ideally, when the character follows a given path very well and thus is directly above it, the character's current position will same as $\mathbf{p}_d$.

The raw future query uses the path point $\mathbf{p}(i)$ corresponding to the current frame i as $\mathbf{p}_d$, assuming that the character follows the path well. In this manner, however, the character cannot follow the path point quickly enough, especially when the user specified a path which is too fast. In such a case, all the path points can be exhausted too early (see Section VIII-F for the results of this experiment).

To prevent premature exhaustion of path points and to effectively use all the path points, we solve the following optimization problem to find $\mathbf{p}_d$ $(= \mathbf{p}(i_d))$ that best matches the character's current position:

$$i_d = \underset{i}{\operatorname{argmin}} \|\mathbf{p}_c - \mathbf{p}(i)\| \quad \text{s.t.} \quad i_d^{\text{prev}} - \alpha \le i \le i_d^{\text{prev}} + \beta. \tag{1}$$

Here, $\mathbf{p}_c$ is the current position of the character root projected to the ground plane, and i_d^{prev} is the frame number corresponding to the desired position obtained in the previous frame. The search range is restricted through two constants α $(= 0)$ and β $(= 10)$, assuming that the desired position can be chosen from the points in the very near future without returning to a previous point in the past. Because the search range is narrow and the calculation is simple, a brute force method calculates and compares all values. For reference, if $\alpha = i_d^{\text{prev}}$, $\beta = i^{\text{last}} - i_d^{\text{prev}}$ are used here, this method chooses the path point closest to the character's position as $\mathbf{p}_d$ (i^{last} is the frame of the last path point in the input path). A comparison with this scheme can be seen in Section VIII-F.

B. SPEED LIMIT FOR FUTURE POSITIONS

Given the character's current position $\mathbf{p}_c$ and the corresponding desired position $\mathbf{p}_d$ on the input path and its frame i_d, a query needs to be generated so that the character follows the future path after frame i_d. The raw future query has the three future path points $\mathbf{p}_f^1 = \mathbf{p}(i_f^1)$, $\mathbf{p}_f^2 = \mathbf{p}(i_f^2)$, $\mathbf{p}_f^3 = \mathbf{p}(i_f^3)$ such that $i_f^1 = i_d+10$, $i_f^2 = i_d+20$, and $i_f^3 = i_d+30$. However, in this way, if the user draws a path too fast, the gap between the three future path points becomes too wide, resulting in a query that is not similar to any of the future queries stored

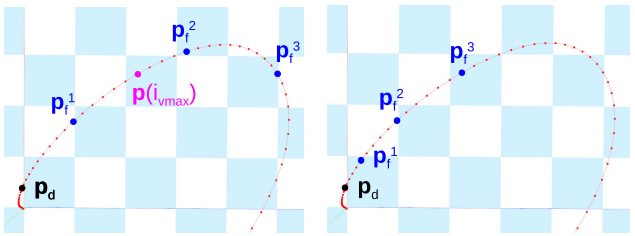


FIGURE 3. Applying a speed limit to future positions. (a) Example of a case where the initial value of the last future path point $\mathbf{p}_f^3 = \mathbf{p}(i_d + 30)$ exists after the path point $\mathbf{p}(i_{\text{vmax}})$ reachable with the maximum moving speed (i.e. $i_d + 30 > i_{\text{vmax}}$). (b) Adjusted future path points $\mathbf{p}_f^1$, $\mathbf{p}_f^2$, $\mathbf{p}_f^3$ to account for the maximum travel speed.

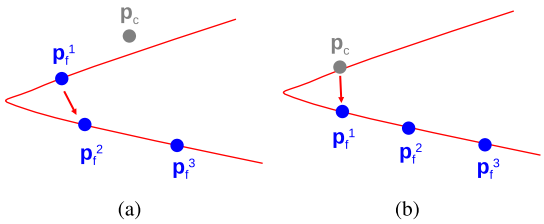


FIGURE 4. An example of future positions in a motion matching query when the input path (red curve) has a sharp corner. (a) a situation where a query is generated such that the character moves downwards following the red arrow at the point $\mathbf{p}_f^1$ in the future. (b) a situation in which a query is generated such that the character moves downwards following the red arrow from the current position of the character. In both cases (a) and (b), the character will likely not follow the path well around the corner.

in the feature database, which in turn leads to a low quality motion.

To solve this problem, we set the maximum movement speed $v_{\max}$ of the character, and restricts the possible locations of the future path points $\mathbf{p}_f^1$, $\mathbf{p}_f^2$, and $\mathbf{p}_f^3$ accordingly. Specifically, let $\mathbf{p}(i_{\text{vmax}})$ be the path point which is reached when the character moves at speed v_{max} for 30 frames. Then, if the last future path point $\mathbf{p}_f^3$ in the query is further away from $\mathbf{p}_d$ than $\mathbf{p}(i_{\text{vmax}})$ along the input path, $\mathbf{p}_f^3$ is changed to $\mathbf{p}(i_{\text{vmax}})$. Also, $\mathbf{p}_f^1$ and $\mathbf{p}_f^2$ are resampled at uniform time intervals, that is, $\mathbf{p}_f^1 \leftarrow \mathbf{p}(i_d + (i_{\text{vmax}} - i_d)/3)$ and $\mathbf{p}_f^1 \leftarrow \mathbf{p}(i_d + 2(i_{\text{vmax}} - i_d)/3)$. Otherwise, $\mathbf{p}_f^1$, $\mathbf{p}_f^2$, and $\mathbf{p}_f^3$ are all kept as their original values at intervals of ten frames (Figure 3).

$v_{\max}$ can be determined by referring to the movement speed distribution of the motion database, and this way, even when a specific user draws a path at an excessively high speed, the future path query is always within the range observed in the feature database, and the quality of the resulting motion is maintained.

C. MODIFYING FUTURE POSITION FOR SHARP CORNERS

When there is a sharp corner in the user-drawn path, the raw future query is often unable to generate motion so that the character follows the path well around sharp corner. This occurs when the query does not contain the path point near the sharp corner due to the temporal gap between samples in the query (Figure 4). In other words, this is because the future positions of the query are sampled at an interval greater than one frame. However, if $\mathbf{p}_f^1$, $\mathbf{p}_f^2$, $\mathbf{p}_f^3$ are sampled at one frame interval, the entire feature database must be updated to use the same frame interval instead of the default intervals of ten frames, and the quality of the resulting motion will not be good because matching is performed considering only the very near future. Therefore, it is necessary to generate a query by modifying the future positions to advance further toward the corner vertex rather than to reduce the temporal gap. Another problem that arises with sharp corners is that the character often gets stuck near the corner without moving forward any further. This is because a query is generated to change the movement direction before the character reaches the corner vertex, so the frame i_d calculated by Equation 1 can no longer advance forward along the input path.

We propose a simple heuristic method to tackle these problems. We first determine whether a sharp corner exists within the interval containing the desired position $\mathbf{p}_d$ and the three future path points $\mathbf{p}_f^1$, $\mathbf{p}_f^2$, $\mathbf{p}_f^3$ which are computed as described in Section VI-B. If so, a new query is created to make the character moves further in the corner direction; the position of the future path point after the sharp corner is adjusted so that it is on the line connecting the previous future path point and the path point just before the corner. Please refer to Appendix D for the details of future position modification for sharp corners.

D. COMPUTING FUTURE FACING DIRECTIONS

In order to compose a motion matching query, not only the positions but also the facing directions of the character need to be specified at the three future frames. Unlike future positions computed from the input path, future facing directions can be generated in various ways. We propose three methods to calculate facing directions.

E. TANGENTIAL DIRECTION

In this method, the facing direction at frame i is determined using the tangential direction vector $\hat{\mathbf{v}}(i)$ of the input path defined as follows:

$$\hat{\mathbf{v}}(i) = \frac{\mathbf{p}(i+1) - \mathbf{p}(i)}{\|\mathbf{p}(i+1) - \mathbf{p}(i)\|}. \tag{2}$$

Using this simple approach, the character always faces forward while moving.

F. JOYSTICK DIRECTION

In this method, we set the facing directions of three future frames using the user's real-time joystick input. When the joystick stays in place or in the 12 o'clock position, the directions calculated using the *Tangential Direction* method are used. If the stick is pushed in the direction away from 12 o'clock by θ, the *Tangential Directions* plus θ are used as the facing directions.

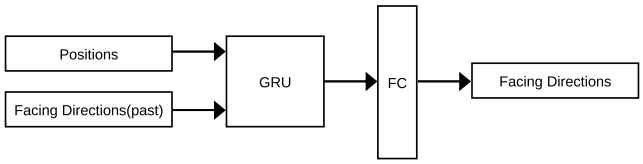
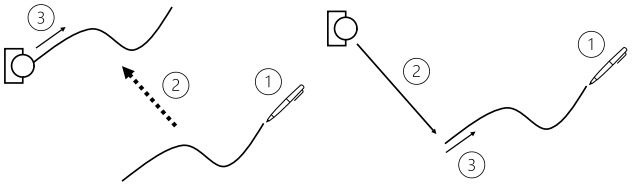


FIGURE 5. The structure of *DirectionNet*.



(a) *Local Control* (b) *Global Control*

FIGURE 6. Conceptual depiction of our interactive control modes.

G. DirectionNet

As the third method, we propose an RNN-based network, *DirectionNet*, that takes the user-drawn path and then infers the natural facing direction of the character at each moment of the path as it would have appeared in our motion database. The *DirectionNet* consists of a 1-layer gated recurrent unit (GRU) [46] network with 30 hidden units followed by a linear layer to produce a series of facing directions from a series of path points during a time window around $\mathbf{p}_d$ (Figure 5). At each time step, it receives the position of each path point and previous facing direction and predicts facing direction at the path point. It is inspired by the *pace network* of QuaterNet [8], but their network takes the trajectory curves and its average speed as an input and outputs the trajectory's amplitude, foot frequencies, and facing directions. Our *DirectionNet* has a simpler network structure and input / output formats; it takes the trajectory and past facing directions and outputs the facing directions only.

To prepare the dataset for the *DirectionNet*, each motion in our motion database is sliced into 4-second segments (120 frames at 30 fps) for both the trajectories for horizontal root position and its facing direction. At runtime, our *DirectionNet* predicts 2-second future facing directions by taking 4-second past and future path points in the user-drawn path. Then three direction vectors at $\mathbf{p}_f^1$, $\mathbf{p}_f^2$, and $\mathbf{p}_f^3$ are picked from the output future facing directions to construct the future facing directions $\mathbf{d}_i$ in the query.

VII. INTERACTIVE CONTROL

In an interactive application, whenever the user has finished drawing a path, the user-drawn path is preprocessed first (Section V). LHMM is performed (Section IV) when the path preprocessing is complete and every N frames thereafter, while generating a query internally (Section VI). Please refer to Appendix B to see how the path-following query generation process is combined with the LHMM algorithm.

For this interactive character path-following task, we propose two control modes described below (see Figure 6 for the conceptual depiction of these two modes).

In *Local Control* mode, the user can draw a path on any visible part of the ground plane. As soon as the path drawing is finished, the path is translated so that its starting point matches the current character position $\mathbf{p}_c$. And immediately, LHMM is performed at the beginning of the translated path, smoothly followed by the next LHMM executed at every N frames. This mode is advantageous for specifying the movement of the character starting from its current location.

In *Global Control* mode, input paths remain at the original positions on the ground as drawn by the user. After the user finishes drawing a path, a line segment connecting the current position of the character and the starting point of the raw input path is concatenated with the input path, and this extended path is smoothed as described in Section V-B. Then, LHMM is regularly performed starting from the beginning of the extended path. The speed of the extended part is set to the average speed during the first second of the user-drawn path. As a result, the character first moves to the start position of the path drawn by the user, and then follows it. This mode is useful when controlling the character to pass through specific locations in the global space. Section VIII-E shows various scenarios using these two control modes.

VIII. RESULTS

In this section, we present various experiments that demonstrate the effectiveness of our methods. Please see the accompanying video for animation results.

Based on motion matching, our system can conveniently generate path-following animations using different types of motions for the same input path by simply changing the motion and feature databases. To demonstrate these characteristics, we construct a motion and feature database from four motion datasets listed below. Each dataset contains motion clips resampled at 30 Hz because motion generation is also performed at 30 Hz in our system.

Locomotion
: This dataset consists of general locomotion excluding jumps and t-poses from the motion capture dataset used in [7], and is about 39 minutes long.

Salsa Dance
: This dataset consists of the motions of subjects #60 and #61 labeled 'salsa' in the CMU motion capture database [47]. It's about 9 minutes long.

Indian Dance
: This dataset consists of the motions of subject #94 labeled 'indian dance' in the CMU mocap database, and is about 7 minutes long.

Basketball
: This dataset consists of the motions of subjects #6 and #102 labeled 'basketball' in the CMU mocap database, and is about 3 minutes long.

Locomotion is used in most experiments for observing the changes in speed given a path or comparing the result motions according to various settings. In the experiments of

Section VIII-D and VIII-E, other datasets are also used for generating various motions for various paths.

Training the *DirectionNet* (Section VI-D) takes about 10 hours for 1000 epochs on a Nvidia GeForce RTX 2070 GPU. The network is updated with L1Loss and Adam optimizer. Since a recurrent network always requires a-step-before input, initial position is replicated as padding when there is no past trajectory log at the beginning.

A. COMPARISON BETWEEN LHMM AND BMM

We compared the results of path following with LHMM and BMM for the same set of paths, *Path A* and *Path B* (Figure 7). The difference is more noticeable in the accompanying video.

LHMM was tested with two settings: *LHMM*($k = 3, l = 3$) and *LHMM*($k = 10, l = 3$). BMM was also tested with two settings: *BMM*($k = 1, l = 1$) which is the default setting in which the future part of the query consists of three future frames, and *BMM with longer queries* in which the future part consists of nine future frames.

In the comparison between *LHMM*($k = 3, l = 3$) and *BMM*($k = 1, l = 1$), the latter produces motion with degraded quality, such as unstable path-following motion that moves left and right on the user-drawn path, or a non-smooth change of direction on a smooth curve. On the other hand, since *LHMM*($k = 3, l = 3$) finds the next frame that can produce the optimal result in a more distant future section, it can be seen that the motion generated by *LHMM*($k = 3, l = 3$) draws a smooth root path and stably follows the user-drawn path.

Both *LHMM*($k = 3, l = 3$) and *BMM with longer queries* may be considered as ways to consider a longer future, but the latter produced low-quality motion that did not properly follow the path. This is presumably because there are not many frames in the feature database that are close to the position and direction of the nine future frames specified from the path, so the cloeset frame from the query is still quite different from the query.

There is a trade-off between the quality of resulting motion and execution speed. It can be thought that larger values of k and l will produce better results, which requires more number of kNN_search() calls. However, the execution speed is strongly correlated with the number of kNN_search() calls as shown in Table 1.

We adjusted the parameters k and l through experiments so that a good balance is achieved in this trade-off. Most experiments were performed using $k = 3$ and $l = 3$ unless stated otherwise. In this experiment, it can be seen that *LHMM*($k = 3, l = 3$) and *LHMM*($k = 10, l = 3$) produce no significant difference overall. This means that even if only the top three candidates are searched in each LHMM step, motion with a quality close to the result of searching for much more candidates can be obtained.

B. CHANGES TO A SPEED LIMIT

In our method, natural motions are generated for arbitrarily speed limits, as long as it does not exceed the speed range in the motion dataset (i.e., the speed range that a person can actually move).

TABLE 1. Statitics for LHMM and BMM (for Path A in Figure 7). The 'Avg. time' and '# of kNN_search' represent the time and the total number of kNN_search() calls for a single top-level LHMM call, respectively.

Configuration	Avg. time (s)	# of kNN_search
LHMM($k = 3, l = 3$)	0.019	13
BMM($k = 1, l = 1$)	0.003	1
BMM with longer queries	0.009	1
LHMM($k = 10, l = 3$)	0.115	111

The histogram in Figure 13 shows the distribution of movement speed in the *Locomotion* dataset used in our experiment. As mentioned in Section VI-B, the user can set v_{max} to limit future path positions referring to this speed distribution. Figure 9 shows the resulting motions generated using various v_{max} values for the same input path.

If v_{max} is set to 1.0 m/s, natural motion transitions are generated by matching a large number of motion frames walking at a speed of about 1.0 m/s, but there is a limitation that only a slow walking motion can be generated (Figure 9 (a)). We found that setting v_{max} to a larger value (= 3.0 m/s) creates a character moving at various speeds and provides overall better user control and the diversity of the resulting motions. There was a concern that the motion quality could be deteriorated because the number of motion frames corresponding to this value in our motion database is small. However, in practice, it is observed that, even when the distance between future positions of the feature is slightly less than that of the query calculated using 3.0 m/s, other features elements are often close enough to the corresponding query elements and can generate a natural transition through motion matching (Figure 9 (c)).

However, when we choose v_{max} much higher than the maximum movement speed observed in the dataset, the character is not able to follow the input path (Figure 9 (d)). This is expected because there are no motion frames in the database containing a future path position that far away. Instead, the character simply try to go straight by following frames with distant future path points ($\mathbf{p}_f^1, \mathbf{p}_f^2, \mathbf{p}_f^3$).

C. FACING DIRECTION CONTROL

Here, an experiment is conducted to calculate the future facing direction of a query in three ways: using *Tangential Direction*, *Joystick Direction*, and *DirectionNet* described in Section VI-D. Using these methods, it is possible to create motions in different styles on the same input path (Figure 10). The *DirectionNet* method can generate not only the gait that moves forward, but also other gaits such as side steps and back steps depending on the given path. The result of *Joystick Direction* can be seen in the accompanying video.

For reference, in the experiments of Section VIII-B and VIII-F, only *Tangential Direction* method is used because these experiments are for observing the differences in the resulting motions according to various settings, and the facing direction can be easily predicted from the given path.

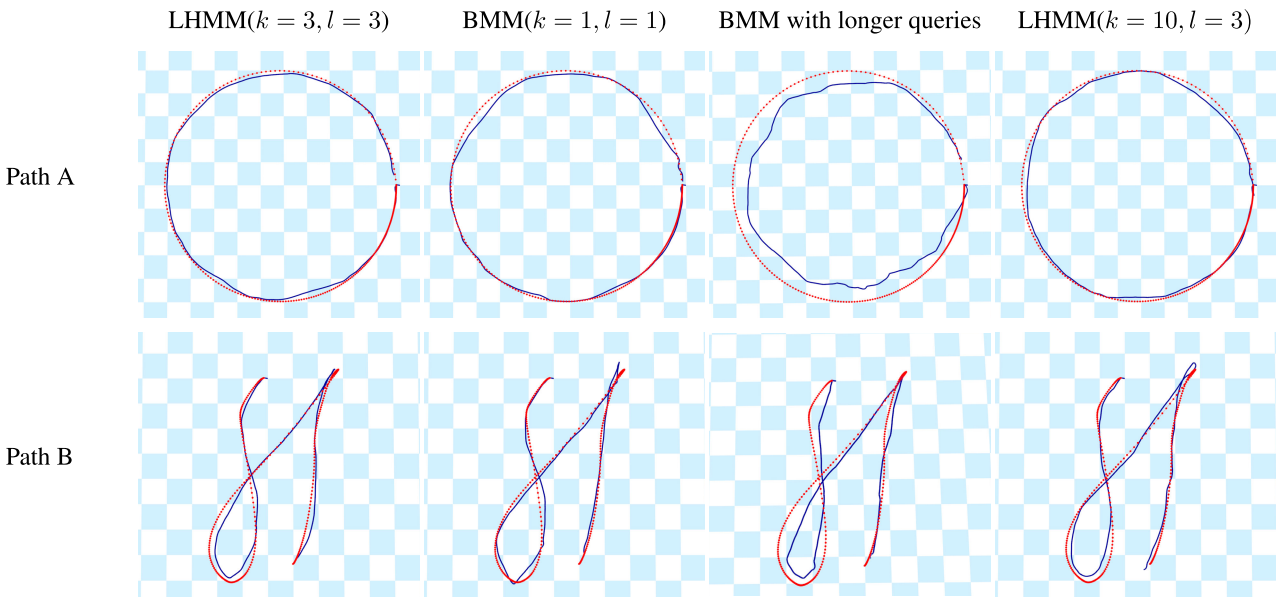


FIGURE 7. Comparison between LHMM and BMM. Red: the user-drawn path. Blue: the root path of the character.

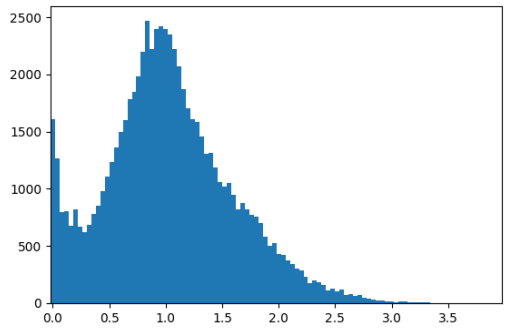


FIGURE 8. The histogram of character movement speed in *Locomotion* dataset. The horizontal axis is the speed in m/s, and the vertical axis is the number of motion frames.

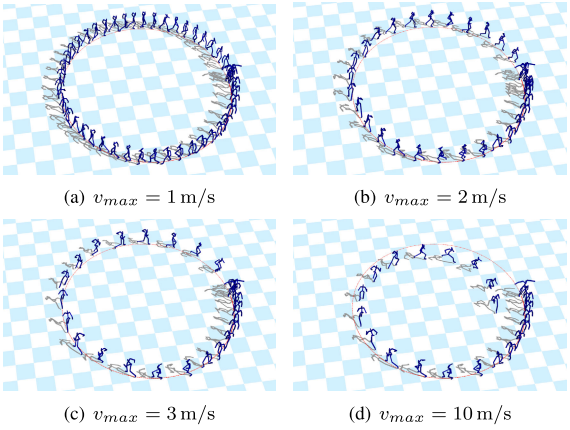


FIGURE 9. The result motions with various v_{max} values for the same input paths. The circular input path was created programmatically to have a linearly increasing speed from 0 m/s to 9 m/s for one and a half laps. The character poses are shown for only one lap of the path for visibility .

D. USING VARIOUS MOTION DATASETS

In this experiment, motions following various input paths are generated using four different motion datasets: *Locomotion*, *Salsa Dance*, *Indian Dance*, and *Basketball*. As shown in Figure 11, each dataset creates an animation with a different style of motion.

E. INTERACTIVE CONTROL DEMOS

To show the capability of our system in terms of interactive control, we design the following scenarios (Figure 12).

Free Move
: When the user freely draws an input path on the ground, the character responds immediately and a path-following motion is created. It operates in *Local Control* mode.

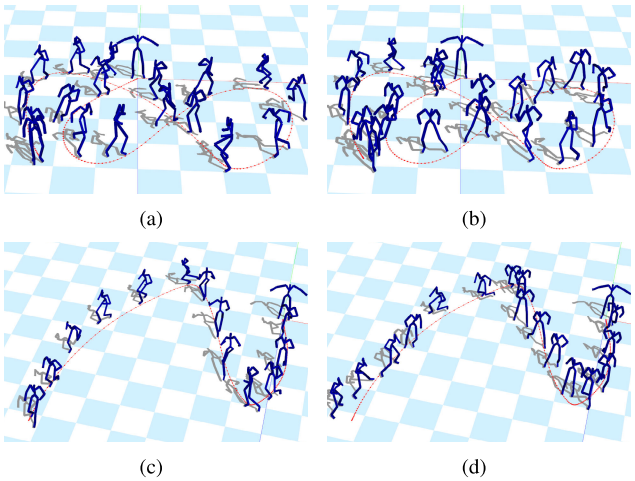
Cross the Finish Line
: The user controls the character to pass the finish line, avoiding obstacles approaching to the right with different speeds. It operates in *Local Control* mode.

Reach the Target Location
: The user controls the character to reach the target location, avoiding static obstacles. It operates in *Global Control* mode.

Multi-Character
: The user controls multiple characters concurrently to reach the target location while avoiding obstacles. The user can change the currently selected character by pressing a key on the keyboard and draw a path for that character. It operates in *Global Control* mode. Since this demo needs to perform LHMM for multiple characters at the same time, a setting of $k = 1, l = 1$ was used to reduce computation time and enable interactive rate.

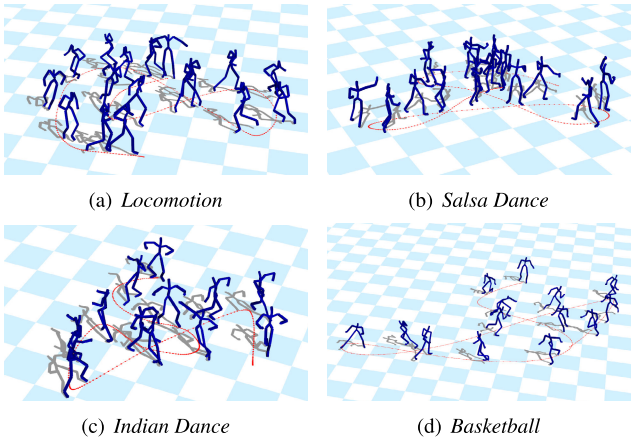
F. ABLATION STUDY FOR FUTURE QUERY REVISION

In this experiment, an ablation experiment is performed for each component of the proposed method: input path



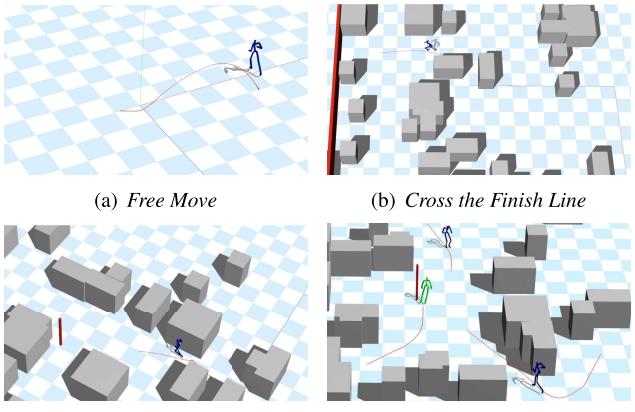
(a) (b) (c) (d)

FIGURE 10. The result motions generated by *Tangential Direction* and *DirectionNet* schemes for the same input paths. (a), (c) *Tangential Direction*. (b), (d) *DirectionNet*.



(a) *Locomotion* (b) *Salsa Dance* (c) *Indian Dance* (d) *Basketball*

FIGURE 11. The path-following motions generated using different motion datasets.



(a) *Free Move* (b) *Cross the Finish Line* (c) *Reach the Target Location* (d) *Multi-Character*

FIGURE 12. Our interactive control demos.

smoothing (Section V-B), desired position computation (Section VI-A), speed limit (Section VI-B), and modification for sharp corners (Section VI-C). Additionally, a comparison experiment is also conducted with the raw future query in which none of the above components is applied (in this method, the path point corresponding to the current frame is used as the desired position $\mathbf{p}_d$). In the experiment, the facing direction is calculated using the *Tangential Direction* method, and the *Locomotion* dataset is used.

TABLE 2. The average distance between the input path and the character root trajectory. '(fail)' means that the character fails to move to the last part of the path ($\mathbf{p}_d$ does not progress to the last path point even after a sufficient amount of time).

Configuration	Average distance (m)			
	Path 1	Path 2	Path 3	Path 4
Proposed (full)	0.1431	0.1471	0.0850	0.1468
No input path smoothing	0.1861	0.1239	0.0887	0.1724
No desired position comp. (current frame path point)	1.1468	0.5228	0.995	0.3267
No desired position comp. (nearest path point)	0.1726	(fail)	(fail)	(fail)
No speed limit	0.4485	0.3279	0.42	0.204
No modification for sharp corners	0.1431	0.1471	0.0850	(fail)
Raw future query	1.045	0.562	1.43	0.328

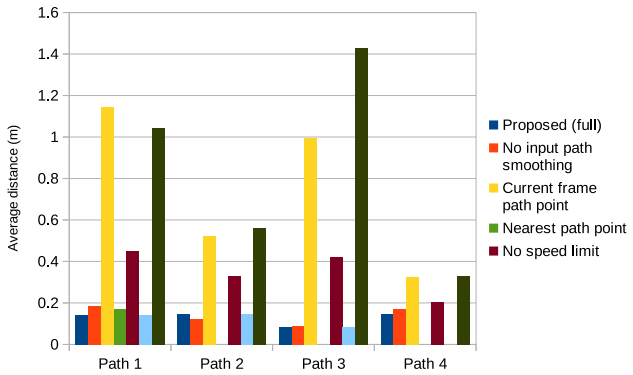


FIGURE 13. Chart for the average distance between the input path and the character root trajectory. Note that the failed tests in Table 2 are not plotted as bars.

When the input path is not smoothed, the movement speed tends to change unevenly in the resulting motion. In particular, the speed changes abruptly at the corner where the direction changes, so the resulting motion is less natural compared to the case where path-smoothing is applied (the second row of Figure 14). This difference is more noticeable in the accompanying video.

If the desired position $\mathbf{p}_d$ is set to the path point corresponding to the current frame and the user draws a path rapidly, $\mathbf{p}_d$ advances too quickly over time, so the character does not follow the path well and sometimes stops in the middle of the input path due to exhaustion of path points (the third row of Figure 14).

If $\mathbf{p}_d$ is specified as the path point on the input path closest to the current character's position, the character often does not follow the path properly or even get stuck in a loop if there is a self-intersection in the input path (the fourth row of Figure 14).

If the future position query is calculated without using the speed limit v_{max}, the character does not follow the curve of path properly when the user draws the path fast (the fifth row of Figure 14).

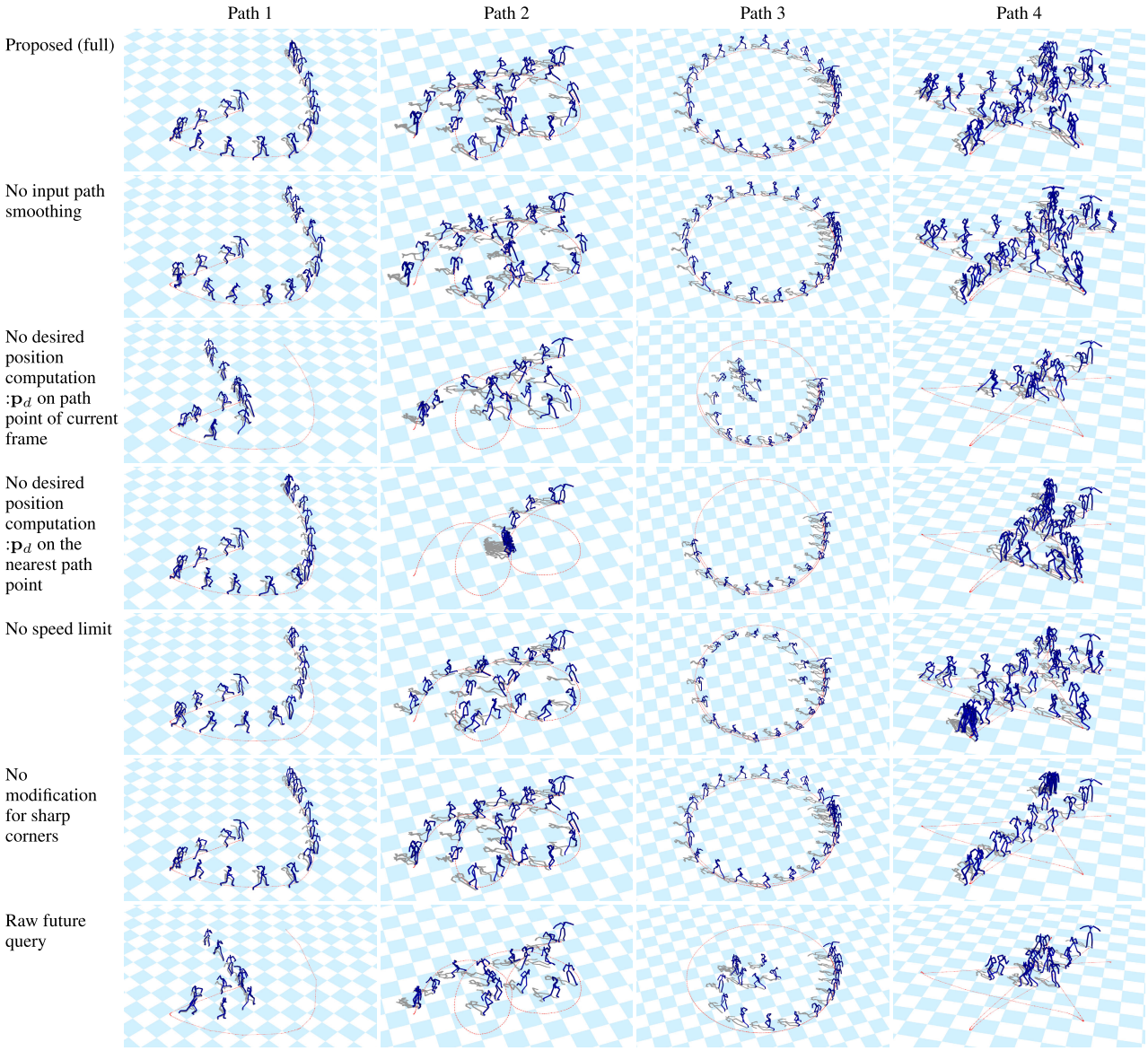


FIGURE 14. Ablation experiments on the proposed components .

If the modification for sharp corners is not applied, the character is often stuck near the corner (the sixth row of Figure 14)). Even when the character gets out of the corner, the character often does not follow the path well around the corner in the resulting motion.

Also, in order to quantitatively compare the path-following accuracy of each resulting motion, we calculate the average distances between the resulting path of the character and the user's input path for the four input paths in Figure 14. The distance between the character path and the input path is calculated by measuring the distance from the character position $\mathbf{p}_c$ for each frame to the nearest path point on the input path and averaging them. The measured distances are summarized in Table 2, and it can be seen that the error is generally the smallest when all of the proposed components are used.

IX. CONCLUSION AND DISCUSSION

We propose an interactive path-following motion generation system based on motion matching. The user can intuitively control the character including its movement speed, in the two interactive control modes *Local Control* and *Global Control*, just by drawing a path on the ground. This is achieved by our long-horizon motion matching algorithm with carefully revised motion matching queries. We also propose three different ways to calculate future facing directions in the query. Our method can be applied to control multiple characters interactively. Additionally, by simply changing the motion dataset used for motion matching, the character follows a user-drawn path with different motion styles.

One of the strengths of our method is that it can reliably generate motion for various path following situations. Path-following may seem like a simple problem, but its application

to real-world interactive applications such as games requires a character to follow the extreme path containing sudden changes in speed or direction drawn by an immersed user in the game. Because our method robustly generates character motion even in such extreme path following situations, it is a reliable method applicable to these real-world interactive applications. Another strength is that our method can be executed efficiently at runtime because it is based on an efficient kNN search. Even though the LHMM does a recursive kNN search, it can run in real-time using enough k and l to produce motion of sufficiently good quality on a typical desktop.

One of the drawbacks of our method stems from the nature of motion matching. Based on motion matching, only motions that exist in the motion database can be reproduced, and this characteristic can be a limiting factor in creating natural motions for various and exceptional input paths. For example, even with input path smoothing (Section V-B) and modifications for sharp corners (Section VI-C), an unnaturally abrupt turning motion can be generated when an overly sharp corner is specified by the user because there is no corresponding motion in the motion database. Another drawback in the current form of our method is that is is only applicable to a character on the plane ground.

One of the reasons motion matching is widely used in the game industry despite advances in deep learning-based motion synthesis is that it can generate relatively high-quality motion in a simple way, using simple features extracted through intuition. This has been demonstrated in [4] and [38], which use the same matching features we used in our experiments. As demonstrated in [43], using learned features would bring other advantages in motion generation, but the current choice of the feature works effectively for our interactive path following tasks without additional computational burden.

In this work, changing the motion style is supported only limitedly by changing the motion dataset that constitutes the motion database. Enabling users to interactively change motion styles could be an interesting future research direction. Improving our method to be applicable to uneven terrain will be an important future work that can make our method more practically useful. It would be interesting to combine the ideas of physics-based approaches or deep learning-based approaches to further improve the generalization capability of our method, without relying solely on reproducing the motions from a motion database.

APPENDIX A
DETAILS OF BASIC MOTION MATCHING (BMM) AND LONG-HORIZON MOTION MATCHING (LHMM)

Both BMM and LHMM requires a motion database containing poses in all motion frames and a feature database containing features for all motion frames.

A feature can be designed in various ways and often relies on human intuition. One example of the feature design is the feature used in our path-following experiments, described in Appendix C. Since each dimension of a feature may significantly vary in magnitude, the feature database for BMM is often normalized using the mean and standard deviation of each dimension. This normalization is also applied to the feature database for LHMM.

The basic idea of BMM is to search for the next frame with feature $\mathbf{x}_j$ closest to the query $\hat{\mathbf{x}}$. When given the query $\hat{\mathbf{x}}$, the goal is to search for the best frame j^* where the Euclidean distance between the $\hat{\mathbf{x}}$ and $\mathbf{x}_j$ are the smallest throughout the feature database:

$$j^* = \underset{j}{\operatorname{argmax}} \|\hat{\mathbf{x}} - \mathbf{x}_j\|^2. \tag{3}$$

Because naive feature-to-feature comparisons are too slow for the requirement of an interactive application, a KD-Tree is commonly used to store precomputed features to accelerate this k-nearest neighbor (kNN) search process (with $k = 1$).

LHMM also uses a kNN search process, typically with $k > 1$. A KD-Tree is also used for kNN search in our LHMM implementation.

In both BMM and LHMM, the matching is performed at every N frames. If the best frame is found, the character motion jumps to that frame and then plays consecutive frames in the motion database until the next matching time. Motion editing techniques such as blending or inverse kinematics are often used at each matching time for smooth transition. In our LHMM implementation, the matching is performed every $1/6$ seconds ($N = 5$ for our 30 Hz motion datasets) with simple motion stitching for smooth transition. We additionally apply an analytic two-joint inverse kinematics to avoid foot sliding artifacts.

APPENDIX B
IMPLEMENTATION OF COMPUTE_QUERY() AND UPDATE_CONTEXT() FOR PATH-FOLLOWING

Algorithm 2 and 3 describe compute_query() and update_context() for our future query revision process (Section VI), which are used in our LHMM algorithm (Algorithm 1).

APPENDIX C
DETAILS OF MATCHING FEATURES

We use the feature design for locomotion proposed in [5] which encodes a full-body posture in each frame into

Algorithm 2 compute_query() for Path-Following

function compute_query_pathfollowing(*pose*, *ctx*)
  ▷ *ctx.pps*: the path points on the user-drawn path, representing $\mathbf{p}(\cdot)$.
  $i_d \leftarrow$ compute_desired_position(*pose*, $ctx.i_d^{prev}$, *ctx.pps*)
  $\{\mathbf{p}_f^i\}_{i=1}^3 \leftarrow$ compute_future_positions_with_speed_limit(i_d, *ctx.pps*)
  $\{\mathbf{p}_f^i\}_{i=1}^3, \{i_f^i\}_{i=1}^3, i_d \leftarrow$ modify_sharp_corner($\{\mathbf{p}_f^i\}_{i=1}^3$, i_d, *ctx.pps*)
  $\{\mathbf{d}_f^i\}_{i=1}^3 \leftarrow$ compute_future_directions($\{i_f^i\}_{i=1}^3$, *ctx.pps*, *ctx.dir_input*)
  $ctx.i_d \leftarrow i_d$
  return create_query(*pose*, $\{\mathbf{p}_f^i\}_{i=1}^3$, $\{\mathbf{d}_f^i\}_{i=1}^3$)
end function

Algorithm 3 update_context() for Path-Following

```
function update_context_pathfollowing(ctx)
    next_ctx ← ctx
    next_ctx.i_d^prev ← ctx.i_d
    return next_ctx
end function
```

a 27-dimensional feature vector:

$$\mathbf{x}_i = \{\mathbf{p}_i^L, \mathbf{v}_i^L, \mathbf{p}_i^R, \mathbf{v}_i^R, \mathbf{v}_i^{root}, \mathbf{t}_i, \mathbf{d}_i\} \in \mathbb{R}^{27}. \quad (4)$$

Here, $\mathbf{p}_i^L, \mathbf{p}_i^R, \mathbf{v}_i^L$, and $\mathbf{v}_i^R$ represent the positions and velocities of the left and the right foot at frame i, respectively. $\mathbf{v}_i^{root}$ is the velocity of the root (the pelvis). The remaining features represent the future path at three different future frames, consisting of the two-dimensional root positions $\mathbf{t}_i \in \mathbb{R}^6$ projected onto the ground and the character facing directions $\mathbf{d}_i \in \mathbb{R}^6$. All the positions and vectors are represented with respect to the character frame, with the origin as the location where the root's position is projected onto the ground, one axis in the front of the root and the other in the global vertical direction.

APPENDIX D
DETAILS OF FUTURE POSITION MODIFICATION FOR SHARP CORNERS

Please see Figure 15 for this process. Here, $\mathbf{p}_d$ will be referred to as $\mathbf{p}_f^0$ for convenience.

For increasing $j \in \{0, 1, 2\}$, we calculate the angle between two tangent vectors $\hat{\mathbf{v}}_f^j = \hat{\mathbf{v}}(i_f^j)$ and $\hat{\mathbf{v}}_f^{j+1} = \hat{\mathbf{v}}(i_f^{j+1})$ obtained from the input path, and if this angle exceeds 80 degrees, it is assumed that a sharp corner exists between $\mathbf{p}_f^j$ and $\mathbf{p}_f^{j+1}$ in the path (the function $\hat{\mathbf{v}}(i)$ is defined in Equation 2). In this case, to avoid a situation where the character is caught in the corner, if i_d calculated by Equation 1 is the same as i_d^{prev}, we add an experimentally chosen value c, that is, $i_d = i_d^{prev} + c$ ($c = 2$ in our experiments) to advance $\mathbf{p}_d$ and recalculate i_f^j.

If a sharp corner exists between $\mathbf{p}_f^j$ and $\mathbf{p}_f^{j+1}$, j will be referred to as J. Let i_{corner} be the moment when the angle between tangents $\hat{\mathbf{v}}(i+1)$ and $\hat{\mathbf{v}}(i)$, $\forall i_f^J < i < i_f^{J+1}$, is maximized. Then $\mathbf{p}(i_{\text{corner}})$ becomes the path point just before the sharp corner. Now, the new $(J+1)$-th future path point $\mathbf{p}_f^{J+1\prime}$ is calculated as $\mathbf{p}_f^J + k(\mathbf{p}(i_{\text{corner}}) - \mathbf{p}_f^J)$ which is on the line connecting $\mathbf{p}_f^J$ and $\mathbf{p}(i_{\text{corner}})$. The line parameter k is calculated as $(i_f^{J+1} - i_f^J)/(i_{\text{corner}} - i_f^J)$ so that the distance between $\mathbf{p}_f^{J+1\prime}$ and $\mathbf{p}_f^J$ is the distance traveled from $\mathbf{p}_f^J$ in time $i_f^{J+1} - i_f^J$. After correcting the location of the $(J+1)$-th future path point in this way, each remaining future path points $\mathbf{p}_f^j$ after the sharp corner is translated by the same amount: $\mathbf{p}_f^j \leftarrow \mathbf{p}_f^j + (\mathbf{p}_f^{J+1\prime} - \mathbf{p}_f^{J+1}), \forall J+1 < j \leq 3$.

The future path point is modified in the manner described above while the character starts approaching the sharp corner section, and until the character goes completely past it. During this process, J is changed in the order of $J = 2, 1, 0$. When $J = 0$, i.e. there is a sharp corner between $\mathbf{p}_d$ and $\mathbf{p}_f^1$, the character may continue to move in the same direction even after passing the sharp corner, depending on when the query is created. To improve the responsiveness of the character to the input path, we create a new query as soon as the character passes the vertex of the sharp corner, that is, when i_d becomes $i_{corner} + 1$, and matching is performed to create a motion that faithfully follows the input path after passing the corner.

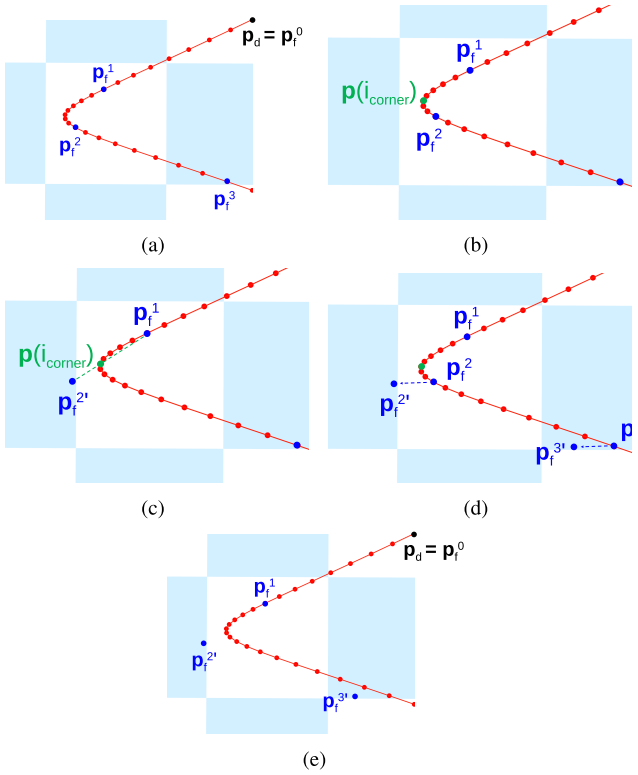


FIGURE 15. Modification of the future positions for sharp corners. (a) A case where a sharp corner exists on the input path. (b) The path point $\mathbf{p}(i_{\text{corner}})$ just before the sharp corner and the future path points $\mathbf{p}_f^1$ and $\mathbf{p}_f^2$ of the query before and after the sharp corner are determined. (c) Calculate the new position $\mathbf{p}_f^{2\prime}$ of the future path point after the sharp corner, $\mathbf{p}_f^2$. (d) Calculate the new positions of the future path points following $\mathbf{p}_f^2$. (e) Final future path points of the adjusted query .

REFERENCES

[1] Imangi Studios, LLC. *Harbor Master*. Gameplay Video. Accessed: Dec. 15, 2022. [Online]. Available: https://youtu.be/41Svfdqo4ek?t=152

[2] Firemint. *Flight Control*. Gameplay Video. Accessed: Dec. 15, 2022. [Online]. Available: https://youtu.be/fZ8kzpWQ0IQ?t=235

[3] H. Zhang, S. Starke, T. Komura, and J. Saito, "Mode-adaptive neural networks for quadruped motion control," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 1–11, Aug. 2018.

[4] D. Holden, O. Kanoun, M. Perepichka, and T. Popa, "Learned motion matching," *ACM Trans. Graph.*, vol. 39, no. 4, p. 53, Aug. 2020.

[5] S. Clavet, "Motion matching and the road to next-gen animation," in *Proc. Game Developers Conf. (GDC)*, 2016.

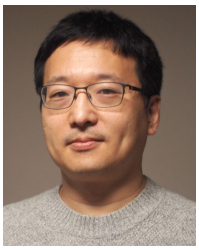
[6] D. Holden, J. Saito, and T. Komura, "A deep learning framework for character motion synthesis and editing," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 1–11, Jul. 2016.

[7] D. Holden, T. Komura, and J. Saito, "Phase-functioned neural networks for character control," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 1–13, Aug. 2017.

[8] D. Pavllo, D. Grangier, and M. Auli, "QuaterNet: A quaternion-based recurrent model for human motion," in *Proc. Brit. Mach. Vis. Conf. (BMVC)*, 2018, pp. 1–14.

[9] H. Y. Ling, F. Zinno, G. Cheng, and M. Van De Panne, "Character controllers using motion VAEs," *ACM Trans. Graph.*, vol. 39, no. 4, p. 40, Aug. 2020.
[10] S. Starke, Y. Zhao, T. Komura, and K. Zaman, "Local motion phases for learning multi-contact character movements," *ACM Trans. Graph.*, vol. 39, no. 4, p. 54, Aug. 2020.
[11] G. E. Henter, S. Alexanderson, and J. Beskow, "MoGlow: Probabilistic and controllable motion synthesis using normalising flows," *ACM Trans. Graph.*, vol. 39, no. 6, pp. 1–14, Dec. 2020.
[12] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 1–10, 2002.
[13] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard, "Interactive control of avatars animated with human motion data," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 491–500, Jul. 2002.
[14] J. M. Wang, D. J. Fleet, and A. Hertzmann, "Gaussian process dynamical models for human motion," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 2, pp. 283–298, Feb. 2008.
[15] S. Starke, H. Zhang, T. Komura, and J. Saito, "Neural state machine for character-scene interactions," *ACM Trans. Graph.*, vol. 38, no. 6, pp. 1–14, Dec. 2019.
[16] Y. Lee, S. Kim, and J. Lee, "Data-driven biped control," *ACM Trans. Graph.*, vol. 29, no. 4, pp. 1–8, Jul. 2010.
[17] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "DeepMimic: Example-guided deep reinforcement learning of physics-based character skills," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 1–14, Aug. 2018.
[18] O. Arikan and D. A. Forsyth, "Interactive motion generation from examples," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 483–490, Jul. 2002.
[19] A. Safonova and J. K. Hodgins, "Construction and optimal search of interpolated motion graphs," *ACM Trans. Graph.*, vol. 26, no. 3, p. 106, Jul. 2007.
[20] O. Arikan, D. A. Forsyth, and J. F. O'Brien, "Motion synthesis from annotations," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 402–408, Jul. 2003.
[21] C. Ren, L. Zhao, and A. Safonova, "Human motion synthesis with optimization-based graphs," *Comput. Graph. Forum*, vol. 29, no. 2, pp. 545–554, May 2010.
[22] R. Heck and M. Gleicher, "Parametric motion graphs," in *Proc. Symp. Interact. 3D Graph. Games*, Apr. 2007, pp. 129–136.
[23] J. Min and J. Chai, "Motion graphs++: A compact generative model for semantic motion analysis and synthesis," *ACM Trans. Graph.*, vol. 31, no. 6, pp. 1–12, Nov. 2012.
[24] H. P. H. Shum, T. Komura, and S. Yamazaki, "Simulating multiple character interactions with collaborative and adversarial goals," *IEEE Trans. Vis. Comput. Graphics*, vol. 18, no. 5, pp. 741–752, May 2012.
[25] J. Lee and K. H. Lee, "Precomputing avatar behavior from human motion data," in *Proc. ACM SIGGRAPH/Eurograph. Symp. Comput. Animation (SCA)*, 2004, pp. 79–87.
[26] A. Treuille, Y. Lee, and Z. Popović, "Near-optimal character animation with continuous control," *ACM Trans. Graph.*, vol. 26, no. 3, p. 7, Jul. 2007.
[27] W.-Y. Lo and M. Zwicker, "Real-time planning for parameterized human motion," in *Proc. ACM SIGGRAPH/Eurographics Symp. Comput. Animation (SCA)*, 2008, pp. 29–38.
[28] M. Buttner, "Machine learning for motion synthesis and character control in games," in *Proc. Interact. 3D Graph. Games (i3D)*, 2019.
[29] F. Zinno, "ML tutorial day: From motion matching to motion synthesis, and all the hurdles in between," in *Proc. Game Developers Conf. (GDC)*, 2019.
[30] G. Harrower, "Real player motion tech in 'EA Sports UFC 3,'" in *Proc. Game Developers Conf. (GDC)*, 2018.
[31] J. Chai and J. K. Hodgins, "Performance animation from low-dimensional control signals," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 686–696, Jul. 2005.
[32] J. Tautges, A. Zinke, B. Krüger, J. Baumann, A. Weber, T. Helten, M. Müller, H.-P. Seidel, and B. Eberhardt, "Motion reconstruction using sparse accelerometer data," *ACM Trans. Graph.*, vol. 30, no. 3, pp. 1–12, May 2011.
[33] K. Grochow, S. L. Martin, A. Hertzmann, and Z. Popović, "Style-based inverse kinematics," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 522–531, Aug. 2004.
[34] S. Levine, J. M. Wang, A. Haraux, Z. Popović, and V. Koltun, "Continuous character control with low-dimensional embeddings," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 1–10, Aug. 2012.
[35] K. Fragkiadaki, S. Levine, P. Felsen, and J. Malik, "Recurrent network models for human dynamics," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 4346–4354.
[36] K. Lee, S. Lee, and J. Lee, "Interactive character animation by learning multi-objective control," *ACM Trans. Graph.*, vol. 37, no. 6, pp. 1–10, Dec. 2018.
[37] S. Starke, Y. Zhao, F. Zinno, and T. Komura, "Neural animation layering for synthesizing martial arts movements," *ACM Trans. Graph.*, vol. 40, no. 4, pp. 1–16, Aug. 2021.
[38] K. Bergamin, S. Clavet, D. Holden, and J. R. Forbes, "DReCon: Data-driven responsive control of physics-based characters," *ACM Trans. Graph.*, vol. 38, no. 6, pp. 1–11, Dec. 2019.
[39] G.-C. Kang and Y. Lee, "Finite state machine-based motion-free learning of biped walking," *IEEE Access*, vol. 9, pp. 20662–20672, 2021.
[40] H. Park, R. Yu, Y. Lee, K. Lee, and J. Lee, "Understanding the stability of deep control policies for biped locomotion," *Vis. Comput.*, vol. 39, no. 1, pp. 473–487, Jan. 2023.
[41] K. Lee, S. Min, S. Lee, and J. Lee, "Learning time-critical responses for interactive character control," *ACM Trans. Graph.*, vol. 40, no. 4, pp. 1–11, Aug. 2021.
[42] K. Chen, Z. Tan, J. Lei, S.-H. Zhang, Y.-C. Guo, W. Zhang, and S.-M. Hu, "ChoreoMaster: Choreography-oriented music-driven dance synthesis," *ACM Trans. Graph.*, vol. 40, no. 4, pp. 1–13, Aug. 2021.
[43] K. Cho, C. Kim, J. Park, J. Park, and J. Noh, "Motion recommendation for online character control," *ACM Trans. Graph.*, vol. 40, no. 6, pp. 1–16, Dec. 2021.
[44] Y. Seol, C. O'Sullivan, and J. Lee, "Creature features: Online motion puppetry for non-human characters," in *Proc. 12th ACM SIGGRAPH/Eurograph. Symp. Comput. Animation*, Jul. 2013, pp. 213–221.
[45] Y. Lee and T. Kwon, "Performance-based biped control using a consumer depth camera," *Comput. Graph. Forum*, vol. 36, no. 2, pp. 387–395, May 2017.
[46] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1–15.
[47] *Carnegie Mellon University Motion Capture Database*. Accessed: Dec. 15, 2022. [Online]. Available: http://mocap.cs.cmu.edu/

JEONGMIN LEE received the B.S. degree in computer science and the M.S. degree in computer software from Hanyang University. She is a Software Engineer with Samsung Electronics. Her research interests include character animation and deep reinforcement learning.



TAESOO KWON received the Ph.D. degree from the Department of Computer Science, Korea Institute of Science and Technology, in 2007. He is currently with the Department of Computer Software Engineering, Hanyang University, Seoul. His research interests include physics-based character animation and machine learning.

YOONSANG LEE received the Ph.D. degree in computer science and engineering from Seoul National University. He is an Associate Professor with the Department of Computer Science, Hanyang University. His research interests include controlling movements of natural or artificial creatures from virtual environment to real world.