




Data-Driven Crowd Motion Control With Multi-Touch Gestures

Yijun Shen¹, Joseph Henry², He Wang³ , Edmond S. L. Ho¹ , Taku Komura² and Hubert P. H. Shum^{1,*} 

¹Northumbria University, Newcastle upon Tyne, United Kingdom
{yi.shen, e.ho}@northumbria.ac.uk

²University of Edinburgh, Edinburgh, United Kingdom
{j.henry, tkomura}@ed.ac.uk

³School of Computing, University of Leeds, Leeds, LS2 9JT, United Kingdom
h.e.wang@leeds.ac.uk

Abstract

Controlling a crowd using multi-touch devices appeals to the computer games and animation industries, as such devices provide a high-dimensional control signal that can effectively define the crowd formation and movement. However, existing works relying on pre-defined control schemes require the users to learn a scheme that may not be intuitive. We propose a data-driven gesture-based crowd control system, in which the control scheme is learned from example gestures provided by different users. In particular, we build a database with pairwise samples of gestures and crowd motions. To effectively generalize the gesture style of different users, such as the use of different numbers of fingers, we propose a set of gesture features for representing a set of hand gesture trajectories. Similarly, to represent crowd motion trajectories of different numbers of characters over time, we propose a set of crowd motion features that are extracted from a Gaussian mixture model. Given a run-time gesture, our system extracts the K nearest gestures from the database and interpolates the corresponding crowd motions in order to generate the run-time control. Our system is accurate and efficient, making it suitable for real-time applications such as real-time strategy games and interactive animation controls.

Keyword: Animation

ACM CCS: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

Controlling a crowd using hand gestures captured by multi-touch devices appeals to the computer games and animation industries. First, multi-touch systems are getting more and more popular nowadays due to the advancement of hardware. Secondly, a crowd has a large degree of freedom, which is difficult to be controlled using traditional controllers with lower dimensional control signals, such as mice and keyboards. Multi-touch devices register several simultaneous control inputs, such that the user can control the complex formation of a crowd intuitively.

The hand gestures captured by multi-touch devices are typically sets of time series of finger positions. Many existing works show that it is possible to map such control signals to a crowd motion using pre-defined control schemes [HSK12, HSK14]. This allows the user to control the formation and movement of the crowd

by performing specific gestures. While these manually designed control schemes are efficient in crowd control, different systems usually employ different control schemes. This is because there are an infinite number of possible mappings between the gesture and the crowd space. Rules need to be explicitly defined to fulfill the control needs optimally. As a result, the users have to learn the schemes in advance before using the systems. Unlikely previous work, we learn a mapping that focuses on both user friendliness and control expressibility in this work to shorten the learning curve.

To this end, we present different crowd motions to a group of users and ask them to give their desirable control gestures, which allow us to generalize the preferred gestures and implement an intuitive control scheme. For every crowd motion in our training data set, we ask the users to perform a control gesture that they think to be the best to create such a motion. It results in a database with pairwise samples of gestures and crowd motions. During run-time, we obtain a gesture from the user, and find the K nearest gestures from the database. We then interpolate the corresponding K crowd motions in order to generate the run-time control. Since the control scheme

*Corresponding author: Hubert P. H. Shum (hubert.shum@northumbria.ac.uk)

is learned from different users without prior constraints, our system is intuitive to use.

One important component of our research is the gesture space representation. As we do not impose any constraints when collecting the control gestures, a representation invariant to individual gesture variations is needed, such as the number of fingers used, different speed, etc. Users often articulate gestures with one or both hands, using multiple fingers when performing similar tasks [RGR13]. At the same time, they show similar variations in their gestures when asked to provide control for the movement of robot groups [MDC*09]. We propose a set of gesture features that effectively represents a wide variety of gestures while independent to inter-user style differences. This includes the centroid feature, the distance to centroid feature, the rotation feature and the minimum oriented bounding box (MOBB) features. We further propose a distance function to evaluate the distance between two gestures in such a space in order to obtain the K -nearest neighbours of a run-time gesture.

Similarly, crowds under different scenarios contain variations such as the numbers of characters within the crowd, and therefore crowd motions also require a general representation. Such a representation should ideally parametrize the whole crowd motion space based on the crowd data. We propose a crowd motion feature space that models a crowd motion with a Gaussian mixture model (GMM), in which the trajectory of each character is modelled by the distribution of the Gaussian component. The major advantage of GMM is that we can set up multiple Gaussian components to accurately model the movement of small groups of characters within the crowd. We further propose a scheme to interpolate multiple crowd motion in the feature space in order to generate the run-time control signal.

We demonstrate that our system can accurately infer the crowd motion based on a given gesture. Users can effectively control a crowd of arbitrary size with intuitive gestures and guide the crowd to navigate through a given virtual environment. Our system is best to be applied in computer games like the crowd control systems in real-time strategy games, and in interactive character animation designs.

This paper presents the following contributions:

- We propose a data-driven method for inferring an appropriate crowd motion based on the gesture input obtained from a touch device. Our approach is not restricted by a pre-specified control scheme. Instead, the control scheme is learned as a mapping between user-preferred gestures and corresponding crowd motions, which encodes both user-friendliness and control expressibility.
- We propose a set of gesture features that are invariant to the variations of the user's preferred touch input style such as the number of fingers used. These features are used for recognizing different properties of a user's multi-touch input, allowing the system to distinguish between a variety of control signals.
- We propose to represent crowd movement with a set of crowd motion features that are obtained from GMM. This representation allows modelling different subgroups of the crowd and is independent of the number of characters. We further propose a

method to interpolate crowd motion features in order to generate a new crowd motion that matches the user input the best.

The rest of the paper is organized as follows. Section 2 discusses the related works and identifies the research gap in gesture-based crowd control. Section 3 provides an overview of our proposed system. Section 4 details the data collection process in order to create the gesture and crowd motion database. Sections 5 and 6 explain our proposed gesture space and crowd motion space, respectively. Section 7 explains the method to synthesize run-time crowd motions. Section 8 gives detailed evaluations on the system. Finally, Sections 9 and 10 discuss the limitations and concludes the work.

2. Related Work

Crowd simulation has been widely used in many areas such as entertainment production and urban planning, where two central issues are control and simulation. Related to our research, there are mainly three subfields where we draw our inspirations upon: gesturing on multi-touch devices, crowd motion control and formation control.

2.1. Gesture recognition on multi-touch device

Since the invention of multi-touch devices, gestures have provided a rich capacity of control input design. Gestures can be sequenced to express complex control purposes and are typically represented by time series of positions and velocities. For any pre-designed stroke patterns, there are some user input variations. Spatially based control design [JTZ*12, VAW12, RVG14] mainly targets on recognizing stroke patterns out of variations to improve the expressibility, but with limited understanding on the time dependencies between strokes. To model temporal or semantic dependencies, rule-based systems such as gesture formalization [GCG10], grammar [GGH*03, KWK*10], state machines [LL12] or syntax [KHDA12] are proposed. However, they either lack the accommodation of user input variations or do not generalize well.

Among the previous research works, Lü and Li's work [LL13] is most related to ours. They present a set of features based on translation, rotation and scaling of a user's finger configurations to encode strokes and recognize gestures. Our proposed gesture representation has similar concepts but different designs for better representativeness. In particular, we utilize both the average distance to centroid and the MOBB, instead of a single scaling parameter, to help identifying gestures such as *expand* versus *split*. Furthermore, instead of simply recognizing the gestures, our system focuses on finding a good mapping between gestures and crowd motions, which involves comparing gestures using the proposed representations. This has not been explored in previous work such as [LL13].

2.2. Crowd motion control

Crowd motion control has been studied extensively, including controlling the whole crowd [Par10, GD11], subgroups [OO09, KHKL09], sets of control points [KLLT08] and the style [LCHL07, JCP*10].

Field-based control focuses on the design of guidance fields in the environment. Dynamic potential fields can be used to represent the flow of the crowd with respect to other moving characters and the environment [TCP06]. Vector fields are typically used to guide each subgroup of the crowd [KSII09]. In [JXW*08], the user can control crowd motion by adding anchor points to indicate their moving directions, with which a vector field is generated. In [PVDBC*11], the movement of the agents is generated by the guidance field that is sketched by the user or extracted from the video. Such a field is used to construct a navigation field that refines the flow of the crowd by avoiding collisions in the environment. While these methods enable the user to easily author the movement of a crowd, they typically require a high-dimensional representation of the crowd motion based on the 2D terrain. Field-based methods are, therefore, ineffective for data-driven crowd control, as the system needs to learn from/interpolate high-dimensional feature vectors. Motivated by the strength of data-driven systems that they can model complex relationships between gestures and crowd motions, we decide to represent the crowd using GMM, in which crowd motions are modelled by low-dimensional feature vectors.

Mesh-based control is another control scheme that evaluates the crowd movement and formation using mesh deformation. Utilizing a single-pass algorithm, crowd movements can be evaluated based on a deformable mesh [HSK12, HSK14]. It is also possible to interactively edit large-scale crowd while maintaining the spatial relationship between individuals [KSKL14]. Voronoi diagram can be used to represent the spatial relationship between different agents and organize the crowd movement in constraint space using Torso Crowd Model [SMTT*17].

One particular problem of existing methods is the lack of high-dimensional control signals that can be used to define the movement details of a crowd that consists of multiple subgroups. Existing methods typically employ multiple levels of control rules, such that the user can define the overall crowd movement first, and define the details of subgroup later. Instead, we decide to embed the control mechanism into our learned mapping between the control signal and the corresponding crowd motions. It solves the problem of potentially contradictory control objectives on different levels, such as different overall crowd and subgroup targets.

2.3. Formation control

Formation control is a technique to control the movement of crowds while maintaining formations. A significant number of papers propose to represent the shape of the crowd by modelling the geometric relations between individual agents. Mesh-based methods are very popular because they can easily represent the formation and accommodate some randomness due to individual motions by controlled mesh deformation. Laplacian mesh editing [SCOL*04] controls and combines existing crowd formations into larger scale crowd animation [KLLT08]. An intermediate 2D mesh between user input and crowd motion can be defined so that crowd formations are controlled by simple user gestures [HSK12, HSK14]. Spectral analysis smoothly transforms the crowd from one formation to another which is represented by Delaunay tetrahedral meshes [TYK*09]. A local coordinate system called formation coordinates maintains the adjacent relationship between individuals in the crowd [GD11]. More

variants of these methods can be found in [KO10, ZZC*14, GD13, XWY*15].

The *Morphable Crowds* [JCP*10], which is based on data examples of different styles of crowd motion, is conceptually similar to our work. While their method is based on modelling the positions of characters surrounding an individual in a crowd motion, our method models the full trajectories of characters in the crowd. Such a full modelling enables us to build up a precise control mapping from the input to crowd motions, which enhances the quality of controlling and synthesizing new crowd motions.

Path pattern that consists of flows of location-orientation pairs is also a good representation of crowd motions, which can be extracted from crowd video [WOO16]. However, the representation is complicated and is too computationally expensive to be used for interactive control purpose.

3. Method Overview

The overview of our system is shown in Figure 1. In the offline stage, we collect user data that describe the mappings between gesture inputs and given crowd motions and create a database. We prepare a number of pre-computed crowd motion trajectories (Figure 1a) and obtain the corresponding gesture trajectories (Figure 1b) from the users. As trajectories information has inconsistent dimensions and inefficient representation, we propose to map gesture and motion trajectories into their respective high-level feature space (Figures 1c and d). The correlated gesture and crowd motion feature spaces generalize and unify the representation of the gesture and crowd data, respectively. In the online stage, our system receives run-time user gestures and evaluate the corresponding crowd motion. Given the run-time gesture trajectories (Figure 1e), we calculate its gesture features (Figure 1f) and conduct a K -nearest neighbours (KNN) search in the database. This allows us to obtain K similar gestures and their corresponding K crowd motion. We interpolate the K crowd motion and generate the resultant run-time crowd motion features (Figure 1g), which is finally converted into crowd motion trajectories (Figure 1h) that can control the run-time crowd. Since the gesture data in the database are obtained from real user inputs with different variations, our system allows intuitive control of crowd in real-time.

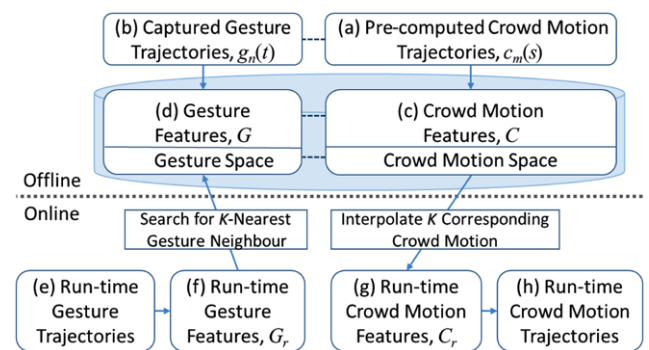


Figure 1: The overview of our crowd control system.

4. Data Collection

In this section, we explain how we collect user gesture data based on a set of pre-generated crowd motion data.

We first generate a set of crowd motions with the crowd simulation system presented in [HSK12, HSK14]. We created 150 motions under 10 different motion classes, which are shown in Figure 2. Such a set of crowd motions consists of six classes of typical crowd motions, including *translate* (i.e. characters all moving in a direction), *twist* (i.e. characters moving in a circular direction around the centre of the scene), *contract* (i.e. characters moving towards the centre of the scene), *expand* (i.e. characters moving away from the centre of the scene), *join* (i.e. two groups of characters moving towards one another) and *split* (i.e. two groups of characters moving away from one another). It also consists of four classes of more complicated crowd motions, including *split then translate*, *translate then join*, *twist while expanding* and *twist while contracting*. The motion set is designed to demonstrate that our system can handle typical crowd motions seen in computer games and movies, as well as complicated motions that consist of combinations of typical motions. Our proposed framework is easily extensible. Developers can add or

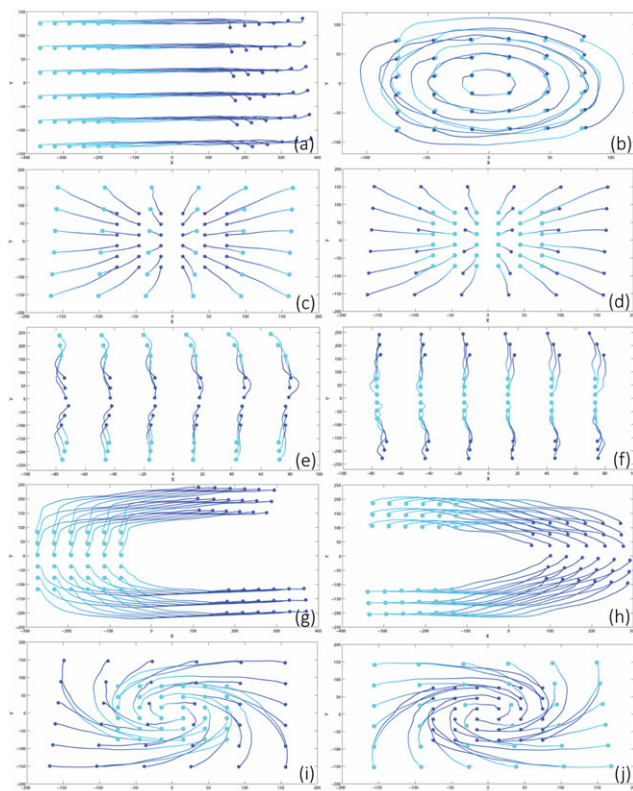


Figure 2: Examples of crowd motion shown to users to collect their control gestures, with the light blue colour indicating the start of the motion and the dark blue indicating the end: (a) *translate*, (b) *twist*, (c) *contract*, (d) *expand*, (e) *join*, (f) *split*, (g) *split then translate*, (h) *translate then join*, (i) *twist while expanding* and (j) *twist while contracting*.

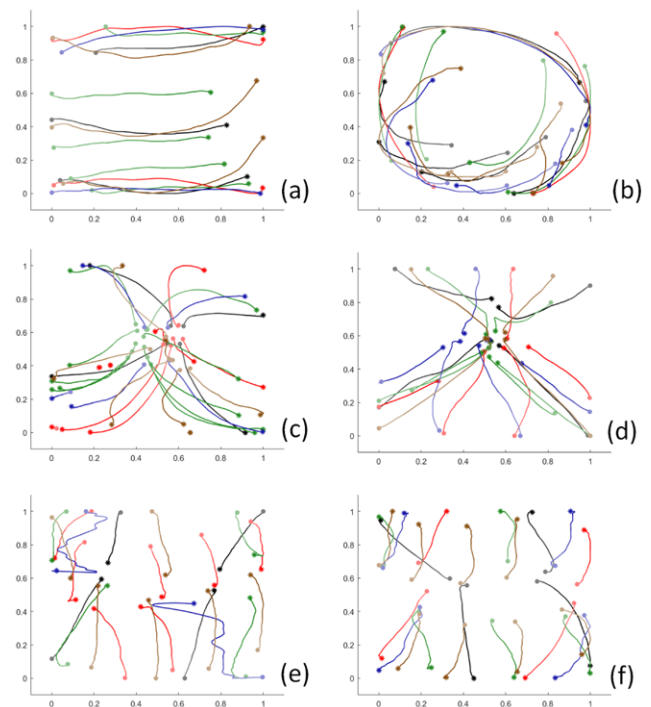


Figure 3: Examples of collected user gesture, with the light colours indicating the starts of the trajectories, the dark ones indicating the ends and different colours representing different set of gestures: (a) *translate*, (b) *twist*, (c) *contract*, (d) *expand*, (e) *join* and (f) *split*.

remove classes of crowd motions based on the requirement of the target application.

Ten volunteer participants, aged between 20 and 50, were asked to provide gestures for the crowd motions shown on a touchscreen (the Wacom Cintiq 27QHD sized 27" diagonally). Each participant was allocated with 15 crowd motions, which were randomly picked from the 150 motions required to train the system. The participants were asked to provide a corresponding hand gesture on the screen as if they were controlling each of such motions with two or more fingers. They were not given any instruction about what the gestures should be, the number of fingers to be used, as well as the duration of the gestures. The orientation of the crowd motion on the screen was varied to prevent any bias in terms of the positioning of the hands when recording the gesture. On average, it took the participants 7 s to observe a crowd motion and provide a gesture. Figure 3 shows some example input of typical crowd motions.

5. Gesture Space

The success of finding a good mapping between the gesture and crowd motion space lies in their corresponding parametrization, such that the variation of the data can be captured. For gestures, we find that the combination of multiple features provide a powerful representation. In this section, we propose a set of gesture features that can be extracted from raw gesture trajectories. Such features form the *gesture space*, which is a low dimensional, continuous space

in which each point represents one gesture. It allows us to compare and distinguish gesture effectively.

Our concept of a gesture space is similar to the idea of *Motion Fields* [LWB*14], in which the authors propose a high-dimensional continuous space that incorporates the set of all possible motion states in character motion. However, unlike character motion, the way a user performs a particular gesture can vary significantly from person to person. For example, users can use a different number of fingers to perform the same intended gesture. Our proposed gesture space consists of a set of features that are independent of such inter-user variations, while capable of capturing the intent of the input. This allows us to robustly distinguish between different types of user gesture.

5.1. Gesture trajectories

Here, we define the representation of raw gesture trajectories and explain the process to resample the gesture with a spline function.

A raw gesture is described by the set of trajectories corresponding to the finger inputs provided by a user. The touchscreen records the position of each touch points in discrete time intervals. As a result, a gesture G_{raw} is defined as a set of trajectories:

$$g_n(t) \quad \forall \quad n \in [1, N], \quad t \in [1, T_n], \quad (1)$$

where N is the total number of trajectories, T_n is the total number of time intervals (i.e. points) in the trajectory n , the representation $g_n(t')$ indicates the 2D location of a specific trajectory n' at a specific time t' . Similar to existing research [WWL07, VAW12, RVG14], we normalize the gesture by translating and uniformly scaling the whole gesture. In particular, the whole gesture is translated such that the minimum x and y position in the gesture is at the origin. We calculate a scaling factor λ to scale the gesture uniformly:

$$\lambda = 1 / \max(d_v, d_h), \quad (2)$$

where d_v and d_h are the maximum vertical and horizontal distance among all points, respectively. After normalization, all trajectory points are within the range $[0, 1] \times [0, 1]$.

We assume all touch trajectories have a similar number of time intervals, as a gesture usually starts and ends with all fingers touching and leaving the screen at the similar time. This allows us to utilize spline functions for approximating and resampling touch trajectories to the same length. In our implementation, we apply the Hermite spline [Lal09] to approximate each of the n touch trajectories. Then, we uniformly resample each trajectory from T_n points to T_H points. The choice of value for T_H is important since undersampling would remove too much information from the original gesture, but oversampling would add unnecessary details and increase computational overhead in later stages. We follow the suggestion in [WWL07] and set $T_H = 64$, which works effectively in all of our experiments. As a result, we define a gesture G as

$$g_n(t) \quad \forall \quad n \in [1, N], \quad t \in [1, T_H], \quad (3)$$

where T is the pre-defined sample number.

There are multiple advantages of approximating and resampling the gesture utilizing Hermite splines. First, different touchscreens have slightly different sample rate. Resampling the trajectories allows the system to work robustly with different hardware. Secondly, it unifies the density of points in a trajectory, which helps us to more accurately identify a gesture using a gesture database. Thirdly, from our discussion with practitioners, crowd control is usually based on the geometry of the trajectories instead of the speed of performing them, as the movement speeds of a crowd are usually constrained in graphics systems. Representing the geometry of the overall trajectories with splines and then uniformly resampling them allow us to model the trajectories with fixed lengths, which removes the speed factor from the trajectories. If the gesture speed is needed, it can be calculated before the resampling stage and stored as an extra feature.

5.2. Gesture features

Here, we define a set of high-level gesture features extracted from the gesture trajectories. Such features are designed to represent the essential components of a gesture in low dimension, making them effective in identifying gestures. Also, they are independent of the number of touch points. As a result, with gesture features, gestures of different touch points can be directly comparable.

The *centroid feature* represents the average position of the user's touch inputs over time. It captures the general shape of the gesture and is independent of the number of touch points. It is defined as a column vector:

$$C(G) = [c_G(1), c_G(2), \dots, c_G(T_H)]^T, \quad (4)$$

where $c_G(t)$ is the centroid at time t :

$$c_G(t) = \frac{1}{N} \sum_{n=1}^N g_n(t). \quad (5)$$

The *distance to centroid feature* represents the distance of each touch point relative to the centroid over time. It allows us to capture the spread of the gesture. It is defined as

$$S(G) = [s_G(1), s_G(2), \dots, s_G(T_H)]^T, \quad (6)$$

where $s_G(t)$ is the spread at time t :

$$s_G(t) = \frac{1}{N} \sum_{n=1}^N |g_n(t) - c_G(t)|, \quad (7)$$

where $|\cdot|$ represents the Euclidean norm, $c_G(t)$ is calculated in Equation (5).

The *rotation feature* represents the average cumulative change in rotation over time of the touch inputs around the centroid. Such a

feature allows us to capture the overall rotation in the gesture. It is defined as

$$R(G) = \left[\sum_{t=0}^0 r_G(t), \sum_{t=0}^1 r_G(t), \dots, \sum_{t=0}^{T_H} r_G(t) \right]^T, \quad (8)$$

where $r_G(t)$ is the average change in rotation at time t :

$$r_G(t) = \begin{cases} 0, & \text{if } t = 0, \\ \frac{1}{N} \sum_{n=1}^N \arctan \frac{(g_n(t) - c_G(t)) \times (g_n(t-1) - c_G(t-1))}{((g_n(t) - c_G(t)) \cdot (g_n(t-1) - c_G(t-1)))}, & \text{otherwise.} \end{cases} \quad (9)$$

The *minimum oriented bounding box feature* represents the minimum and maximum dimension of the MOBB of the touch inputs at each time step. It allows us to represent the movement variation of the gesture over time, which can approximate the area of the gesture. Given a set of touch points at time t , $g_n(t)$, we apply the rotating calipers method [Tou83] to calculate the minimum rectangle bounding the points. We extract the width, $b_w(t)$, and the height, $b_h(t)$ of the rectangle, and define the feature as

$$B(G) = [(b_w(0), b_h(0)), (b_w(1), b_h(1)), \dots, (b_w(T_H), b_h(T_H))]^T, \quad (10)$$

Finally, the gesture space is formed by concatenation of the four gesture features. As a result, a gesture G can be represented by a point in the space with the feature vector:

$$G = [G(G), S(G), R(G), B(G)]^T. \quad (11)$$

5.3. Distance between two gestures

Here, we explain how we compare gestures using gesture features in the gesture space.

Given two gestures G_0 and G_1 , we define the distance as

$$\begin{aligned} D(G_0, G_1) = & \alpha \text{DTW}(C(G_0), C(G_1)) \\ & + \beta \text{DTW}(S(G_0), S(G_1)) \\ & + \gamma \text{DTW}(R(G_0), R(G_1)) \\ & + \delta \text{DTW}(B(G_0), B(G_1)), \end{aligned} \quad (12)$$

where DTW provides a distance between two vectors using dynamic time warping [BC94], and α , β , γ and δ , are weights for each feature. We empirically found that $\alpha = 0.04$, $\beta = 0.36$, $\gamma = 0.36$ and $\delta = 0.24$ work well in our data set. Figure 4 shows two pairs of example gestures, in which (a) and (b) are more different according to Equation (12) ($D = 6.1980$), (c) and (d) are more similar ($D = 0.7064$). This shows that our distance function is less affected by the number of fingers used and is effective in identifying the context of the gestures.

The feature set and distance metrics together determine the well-represented gesture space where algebraic operations can be sensibly defined. It forms the basis of the control scheme learning in later sections.

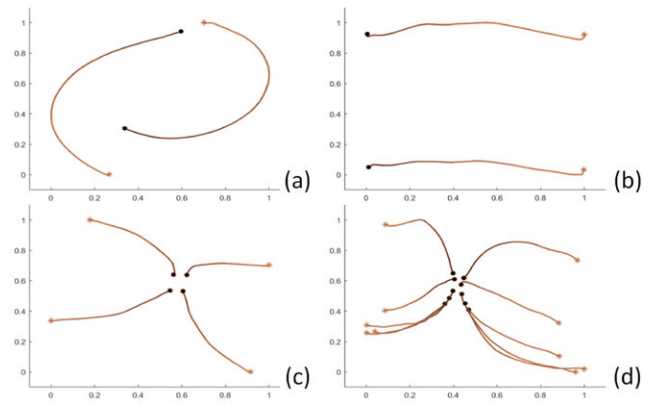


Figure 4: Example gestures, in which (a) and (b) are more different according to Equation (12), (c) and (d) are more similar.

6. Crowd Motion Space

In this section, we present our formulation of a *crowd motion space*, which is conceptually similar to a *gesture space*. We consider the set of movement trajectories from the characters of the crowd, and represent the overall crowd movement with a set of features modelled by a mixture of Gaussian processes.

6.1. Crowd motion trajectories

Here, we represent the motion of a crowd using the trajectories of the characters in the crowd.

A crowd motion C is defined as a set of trajectories:

$$c_m(s) \quad \forall \quad m \in [1, M], \quad t \in [1, S], \quad (13)$$

where M is the total number of character in the crowd, S is the duration of the crowd motion, the representation $c'_m(s')$ indicates the 2D location $(c'_m(s').x, c'_m(s').y)$ of a character m' at time s' . Similar to the gesture trajectories, we normalize the crowd motion trajectories by translating the whole motion such that the starting point is at the origin.

We also resample the crowd motion trajectories from S points to S_H points using the Hermite spline [Lal09] and set $S_H = 64$ [WWL07], as we do for the gesture trajectories in Section 5.1. As a result, a crowd motion C is defined as

$$c_m(s) \quad \forall \quad m \in [1, M], \quad t \in [1, S_H]. \quad (14)$$

For the sake of calculation simplicity, we express the trajectory of the m' th character, $c_{m'}(s)$, as a vector of serialized X and Y positions:

$$\begin{aligned} c_{m'}(s) = & [c_{m'}(1).x, c_1(1).y, c_{m'}(2).x, c_1(2).y, \\ & \dots, c_{m'}(S_H).x, c_{m'}(S_H).y.]^T. \end{aligned} \quad (15)$$

6.2. Crowd motion features

Next, we present our crowd motion features that describe the high-level features of a moving crowd. Such features are independent of the number of characters in the crowd. They can also be used to interpolate two crowd motions in order to generate new ones.

Since the character trajectories in a crowd is controlled by one input gesture, we assume that there exists a linear low-dimensional space that can represent the trajectories of all characters. Trajectories can be treated as functions. Essentially, each crowd motion is a series of 2D functions that defines the trajectories of all characters. This allows us to construct a low-dimensional space and represent the motion trajectories of all characters using Functional Principle Component Analysis (FPCA). FPCA projects a group of functions linearly into a space where a mean function and functional variations serve as the basis of function representation, similar to PCA but on a function level. The mean function, \bar{c}_s where $s \in [1, S_H]$, can be computed by averaging the motion trajectories of all characters. Then, a set of eigenfunctions, E_V^C , and a set of eigenscores, E_S^C can be computed. The eigenfunctions describe the principle movement over time of all characters in the crowd, and the eigenscores represents how the movement of a character can be projected into the low-dimensional space. The trajectory of the m 'th character can be recovered as

$$c_{m'}s = \bar{c}_s + E_V^C E_{S_{m'}}^C, \quad (16)$$

where $E_{S_{m'}}^C$ is the eigenscore of the m 'th character.

Although FPCA gives a compact representation, it does not generalize enough to take all the input variations into account such as different numbers of trajectories or style variations of the same motion. This motivates us to further generalize the representation. We discover that the high-level visual observation of the general motions can be described by the eigenscore distributions. As a result, modelling the crowd motion trajectories of the whole crowd can be considered as modelling the distribution of the eigenscores of the characters. This high-level model allows us to interpolate the distribution of eigenscores, instead of the actual trajectories, between two crowd motions effectively. In addition, such a distribution-based representation does not depend on the number of characters, and does not explicitly map the trajectories of the characters from one crowd to another.

Since the eigenscores of a group of similar motions usually exhibit multi-modality, we propose to use GMM to model the distribution of the eigenscores. There are three main advantages. First, the non-linearity of GMM fits the trajectory data well. Secondly, the multi-modality nature of GMM captures semantic-level meanings such as the crowd being split into multiple groups, which cannot be modelled easily with a single model. This is particularly relevant to crowd motion such as splitting and joining. Finally, multiple GMMs can be easily interpolated and the interpolation has visual as well as semantic meanings, which is important to generate new crowd motions.

There are two import issues in applying GMMs to model the data, which are the optimal parameters and the number of components of the model. We apply the Expectation-Maximization algorithm

[Bis96] to optimize the parameters for the distribution of eigenscores, $\phi(E_S^C)$. The component number essentially allows the system to accurately model multiple subgroups in the crowd motions. In theory, it is possible to automatically determine that by iterating from one and choose the smallest value that reaches the required data likelihood. In practice, we found that users rarely split a crowd into more than two subgroups, even with two hands controlling the crowd. As a result, two Gaussian components are enough to model our database. For simpler motion with only one subgroup, the two components in the GMM blends together to represent the distribution of character trajectories. If more complicated crowd motions with multiple subgroups of characters are involved, an analysis on data likelihood should be performed and more components can be used.

Therefore, the crowd motion features of a crowd C is defined by a vector:

$$C = [\bar{c}_s, E_V^C, \phi(E_S^C)]^T, \quad (17)$$

where \bar{c}_s is the mean trajectory, E_V^C is the set of eigenvectors and $\phi(E_S^C)^T$ is the distribution of the eigenscores modelled by GMM. Conceptually, our crowd motion feature is similar to the *morphable motion primitives* [MCC09, MC12]. The difference is that it is applied on a crowd instead of an individual motion.

Here, we include an optional step to improve the performance of our system. We observe that there is an intrinsic redundancy in the crowd motion trajectories as the characters' trajectories are not arbitrary. Therefore, it is not necessary to use all the eigenvectors E_V^C as the features. In fact, we only use the first 15 principal components returned by FPCA, and the recovered trajectories from Equation (16) achieves <1% error for all the crowd motion in our database. This not only reduces computational cost, but also removes noises that may exist in the motion data.

7. Crowd Motion Control

In this section, we explain how a run-time gesture can be identified based on the set of gestures in the database. Then, we explain how such a gesture generates the corresponding crowd motion.

7.1. Run-time gesture representation

Here, we explain how we represent a gesture using neighbour ones in the database, which allows us to understand the crowd motion the user intended to perform.

We have collected a set of gestures with the corresponding crowd motions as explained in Section 4. The gesture space is non-linear due to the complex nature of hand gesture. Modelling the whole space with high degree non-linear functions would require a large amount of gesture data, which is labour-intensive to obtain and would limit the feasibility to increase the gesture types. Instead, we propose to model a local area of the gesture space that is relevant to the run-time gesture using a linear function. Such a method works robustly even with smaller database and generates reliable results.

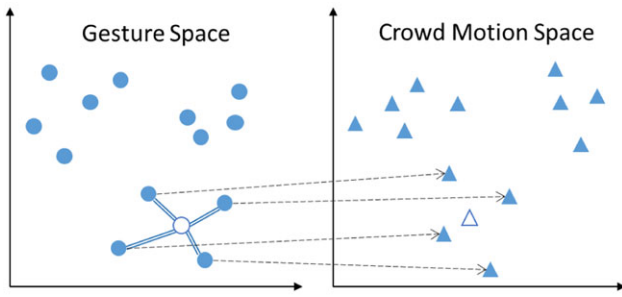


Figure 5: Generating crowd motion with run-time gesture. The circles represent gestures in the gesture space, with the hollow one representing the gesture obtained in run-time. Based on the run-time input, we obtained the K -nearest gestures, visualized by the double lines. The triangles represent crowd motion in the crowd motion space. We find the crowd motions corresponding to the K nearest gestures, pointed by the black arrows. We finally interpolate these crowd motions to create the run-time crowd motion represented by the hollow triangle.

In particular, given a run-time gesture, G_r , we utilize Equation (12) to evaluate its distance with the stored gestures in the database. We represent G_r using a set of K -nearest neighbours, $G_k \forall k = [1, K]$. The neighbours are associated with the corresponding weights, $w_k \forall k = [1, K]$, which are inversely proportional to the distance with respect to the run-time gesture. The particular weight $w_{k'}$ that is corresponding to the gesture $G_{k'}$ is defined as

$$w_{k'} = \frac{1}{D(G_r, G_{k'})} \bigg/ \sum_{k=1}^K \frac{1}{D(G_r, G_k)}, \quad (18)$$

where the summation term acts as a normalization factor to ensure that all the weights sum up to 1.0. In our experiment, we found that $K = 10$ produces good results. This process is visualized in the left part of Figure 5.

Since our gesture database is relatively compact, a brute force search is quick enough to find the K -nearest neighbours in real time. For a larger database, we may organize the database with data structures such as the k -d tree to speed up searching.

The mapping between gestures and motions is necessary to capture the variations of control styles, even for the same motion. A user study could be a good way to establish a mapping but only when there is a consensus on the best gesture for a specific motion. The fact that different users used different gestures even for simple motions suggested that this might not be the case. As an example, for the *twist* motion, some users prefer to use an outwards spiral gesture but some prefer an inwards one. In addition, when doing control on the fly, the input variations are also better handled by the mapping because the input gesture would not be exactly the same as the optimal one, if there is one at all.

While it is possible to apply methods such as regression to evaluate the run-time gesture, we find that KNN is the most reliable way, mainly because our gesture database contains a variety of gestures,

where the sample size is big enough to locally approximate the gesture manifold as hyperplanes. In theory, if the database is dense enough, it could be possible to use the most similar gesture only. However, KNN is more robust against outliers, and constructing a dense database is labour-intensive.

7.2. Run-time crowd motion creation

Here, given the K -nearest neighbours of the run-time gesture, we interpolate the corresponding K crowd motions in the database in order to generate the run-time crowd motion.

Given a run-time gesture, the obtained K -nearest gestures, $G_k \forall k = [1, K]$, are corresponded with K crowd motions, $C_k \forall k = [1, K]$, according to the database. The run-time crowd motion, $C_r = [\bar{c}_s^r, E_V^{C_r}, \phi(E_S^{C_r})]^T$, is evaluated as the weighted sum of the K crowd motions. This process is visualized in the right part of Figure 5. Such an interpolation involves interpolating the crowd motion features individually as follows.

The run-time mean crowd trajectory can be obtained by vector sum, as all mean trajectories in the database has the same size S_H :

$$\bar{c}_s^r = \sum_{k=1}^K w_k \bar{c}_s^k. \quad (19)$$

Similarly, we interpolate and create a new set of eigenvectors:

$$E_V^{C_r} = \sum_{k=1}^K w_k E_V^k. \quad (20)$$

To ensure orthonormalization of the new eigenvectors, we apply the modified Gramm–Schmidt method presented in [CK08].

The blend weights w_k is important to ensure the quality of the resultant GMM, which account for the naturalness of the generated crowd motion. Considering that our gesture-motion pairs in the database are very distinctive and that both the gesture space and crowd motion space can be modelled by the local hyperplane, we use $w_{k'}$ in Equation (18) as the blend weights for the crowd motion w_k . The underlying assumption here is that similar gesture in the gesture space would indicate similar crowd motion in the crowd motion space.

Finally, we propose a mass transport solver based method to combine multiple distributions of eigenscores and generate $\phi(E_S^{C_r})$. A naive one-to-one combination of the Gaussian components of two GMMs does not work well. As shown in Figure 6, assuming each GMM has two components, depending on how we match the components, blending two GMMs has two possible outputs. One of them retains the features from the source GMM, while the other does not as Gaussian components of very different parameters are blended, resulting in a scenario known as cross fading. We follow the displacement interpolation method presented by [BvdPPH11] here. First, given two GMMs, we establish the correspondence of their Gaussian components. Each Gaussian component is defined by a mean value and a covariance value. We evaluate the correspondence using the mass transport solver [HSK12], in which the source

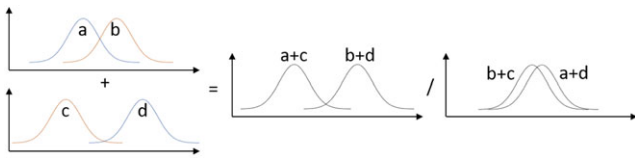


Figure 6: (Left) Mixing two GMM (each with two components) can generate different results depending on how the Gaussian components are matched. (Middle) The desired result that retains the features of the source GMMs. (Right) The undesired result of cross fading.

and target are set as the Gaussian means of the Gaussian components. Secondly, we produce a weighted sum of the Gaussian mean and covariance of each matching Gaussian component, in which the weights are obtained by Equation (18), in order to generate a combined GMM. We iteratively combine all the GMMs in the K -nearest crowd motions, and obtain $\phi(E_S^{C_r})$.

7.3. Crowd motion synthesis

Here, we explain how we apply the crowd motion created in the last section to control a run-time crowd.

Assume that the user is controlling a group of M characters. Given a user gesture, we obtain the corresponding crowd motion $C_r = [\bar{c}_s^r, E_V^{C_r}, \phi(E_S^{C_r})]^T$ as explained in Section 7.2. We first utilize the distribution of the eigenscores, $\phi(E_S^{C_r})$ to sample M eigenscores. Secondly, we apply the eigenscores with the mean trajectory \bar{c}_s^r and eigenvectors $E_V^{C_r}$ to generate M crowd motion trajectories using Equation (16). Third, we apply a mass transport solver [HSK12] to find out the optimal matching between the controlling characters and the calculated crowd motion trajectories. This is done by setting the positions of the characters as the source and the starting points of the trajectories as the target. By using the mass transport solver to evaluate the matching, we avoid visual artefact in which characters have to move a long distance before reaching the starting point of their corresponding trajectories. Finally, the characters move to the starting point of their respective trajectories, and then follow the trajectories, in order to produce the overall motion.

For handling collision detection and avoidance, we implemented the high-level crowd motion synthesis and the low-level character collision avoidance as two separate levels. The high-level system provides the desired position of all characters in the crowd, while the low-level system resolves their positions locally. In our experiments, the low-level system models each character as a cylinder, and utilizes a spring-mass model to calculate the forces required to move the characters into their respective target locations. It resolves the penetration among characters by calculating the push-back forces based on the penetrated depth and direction. Time-integration is applied in each time step to calculate the positions of the characters after all forces are applied. Other more advanced collision avoidance systems can be directly employed into our framework.

We apply the full body motion synthesis method in [SKY12] as an offline process to generate full body running motion based on the point-based movement trajectories. This involves creating a motion

graph that consists of different running actions, and evaluating the optimal action to perform in order to follow the trajectories. We also apply the physical modelling method in [SH12] to create physically plausible movements. This allows us to resolve body part level collisions and penetrations effectively.

8. Experimental Results

In this section, we provide both qualitative and quantitative evaluations of our proposed system.

All the experiments are run with one core of a Core i7 2.67 GHz CPU with 1 GB of memory. For the multi-touch input, we used a G4 multi-touch overlay from PQ labs, attached to a 24" Acer S240HL LCD monitor. In general, the system runs in 40 frames per second, which is higher than the real-time requirement of 30 frames per second. However, there is a slowdown when computing a new crowd motion from a hand gesture, which takes 330 ms, including 300 ms for the KNN searches, 12 ms for generating the crowd motion features, 4 ms for generating and assigning trajectories to characters. We believe that adapting a multi-thread implementation framework can create more consistent frame rate. Also, more efficient search algorithms such as k-d tree search can further reduce the computational time.

8.1. Qualitative evaluations

Here, we evaluate our system qualitatively with different experiments. The readers are referred to our supplementary video for more results.

First, we evaluate the effectiveness of our method by producing a set of crowd motions from a number of user inputs. Figure 7 shows some user gestures and their corresponding simulated crowd motion. Our system generates crowd motions that accurately reflect the different user gesture types. It also works well under different initial positions of the characters. The number of touch points provided for the gestures does not affect our system's ability to produce the appropriate crowd motion.

We setup some virtual environments and ask a user to use our system to control the navigation of the crowd. Figure 8 (upper) shows a corridor environment. The initial crowd cannot fit through the narrow corridor. The user therefore applies a *contract* gesture to reduce the size of the crowd, and two *translate* gestures to move the crowd through the corridor. Finally, the user applies an *expand* to expand the crowd to its original size. Figure 8 (lower) shows a more complicated environment, in which there is an obstacle in the middle of a corridor. The user successively applies the gestures *translate*, *split*, *translate*, *join* and *translate* such that the crowd can avoid the obstacle and reach the other side of the environment. The user finally applies a *twist* gesture such that the crowd can rotate inside the circular environment. These experiments show that our system can potentially be applied to console games that require crowd control, such as the real-time strategy games.

We generate a high-quality, complicated scenario in which 100 characters avoid a number of dynamic moving cars, as shown in Figure 9. The user controls the crowd movement with our

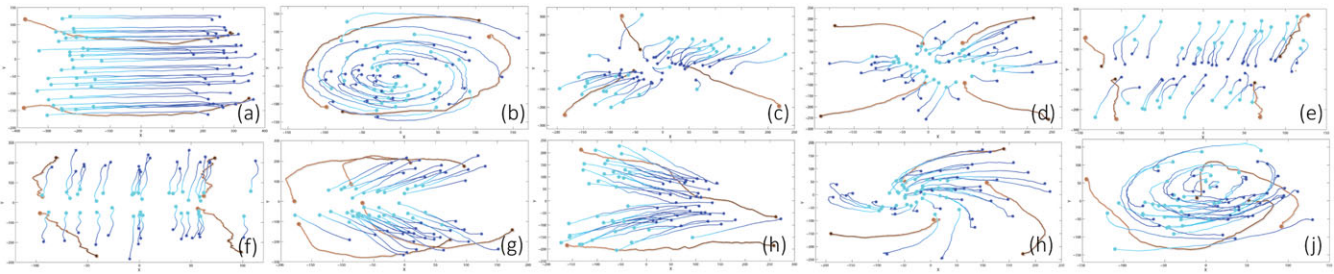


Figure 7: Examples of user input (orange lines) and the corresponding crowd simulation (blue lines) for (a) translate, (b) twist, (c) contract, (d) expand, (e) join, (f) split, (g) split then translate, (h) translate then join, (i) twist while expanding and (j) twist while contracting. The light colours indicate the starts of the trajectories the dark colours indicate the ends.

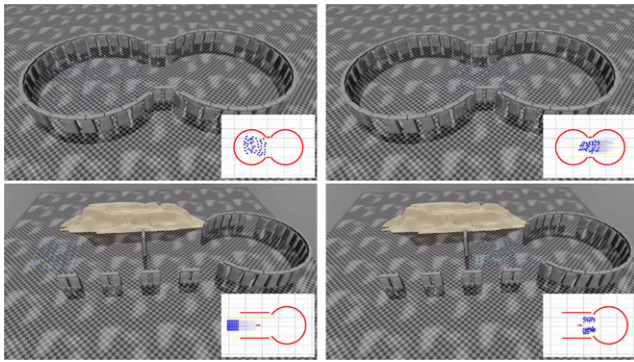


Figure 8: Screenshots of a user controlling a crowd to navigate through (upper) a corridor environment, and (lower) a more complicated environment with an obstacle.

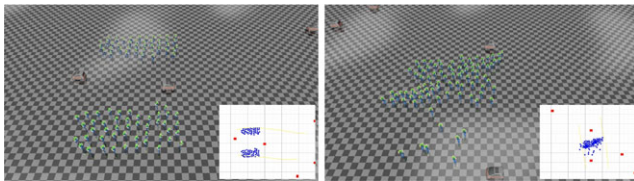


Figure 9: Screenshots of a user controlling a crowd in a complicated scenario with dynamic obstacles.

touch-based system that offers intuitive control on the timing for the change of formation. Multiple gestures are required to steer the crowd. This kind of real-time, precise, interactive control is difficult to be achieved with existing systems. As this demo focuses on demonstrating the animating power of the system for generating realistic scenes, we implement a Gaussian filter to smooth the crowd motion transitions.

While we propose to utilize [HSK12, HSK14] to generate examples for constructing the crowd motions in the database, the overall framework is independent of the underlying method to generate the crowd motions. Basic crowd simulation systems that control characters by setting the starting and goal positions can effectively generate the database and produce comparable results. To demonstrate this,

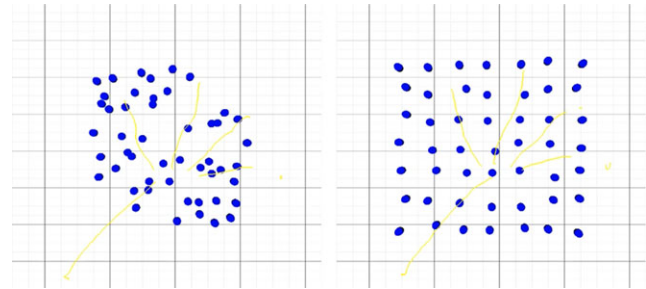


Figure 10: The synthesized expand crowd motion using the database built with (left) Henry et al. and (right) RVO2.

we perform an experiment to utilize the Reciprocal Velocity Obstacle (RVO) 2 system [vdBLM08] to generate the crowd motion database and synthesize new crowd motions based on the user inputs. We compare the newly created results with those generated by our existing database, as shown in Figure 10. We find that while the two databases result in crowds of different behaviour due to the different training data, the resultant crowd motion quality is comparable. This demonstrates the generalizability of our control system.

8.2. Quantitative evaluations

In order to test if our proposed gesture features are discriminative, we conduct leave-one-out cross-validation using the gestures for the six types of typical crowd motions (i.e. translate, twist, contract, expand, join, split). We first use the gesture features of one gesture as testing data for classification, and that of all other gesture as training data. We then obtain the K -nearest gestures. Within them, we conduct a weighted nearest neighbour voting with the weight obtained from Equation (18), where the gesture type with the highest total weight is considered to be the recognized type. We finally check if such a type is the same as the real gesture type of the testing data. We iteratively evaluate all gestures and calculate the average accuracy. Figure 11 shows the confusion matrix of this analysis. It shows that the proposed gesture features are discriminative in order to accurately identify an unknown gesture. The average classification accuracy is 89.6%. For all gesture types except contract, the accuracy is over 87.5%. The contract type has a lower accuracy of 62.5%,

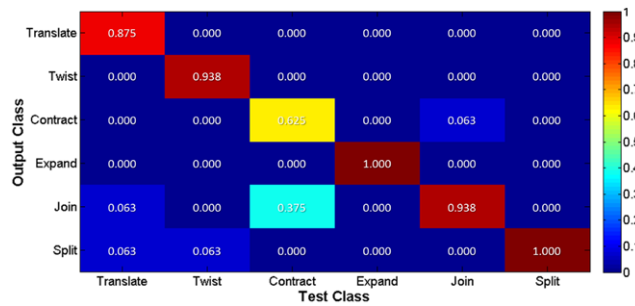


Figure 11: Confusion matrix of six typical motion types. The cell in column i , row j indicates the proportion of all i th test gestures recognized as the j th output gesture.

as some of the gesture samples are very similar to those in the join type.

We visualize the four gesture features in Figure 12 to show that they are effective in representing different gesture types. Here, we group the collected gestures based on the six types of typical crowd motions they represent and calculate the average gesture per group. We then plot the features to see how distinctive different gesture groups can be. While individual features may not be able to clearly represent all the types (e.g. $S(G)$ cannot distinguish expand and split easily), the features are complementary to each other (e.g. $B(G)$ can distinguish expand and split well).

9. Limitations

Our system does not consider the mapping between the gestures and the full-body motions of the characters. Although this is an interesting idea, such a mapping will suffer from the ambiguity such as the walking phase as presented in [HKS17], and extra parameter inputs will be required. Instead, the detailed movements (e.g. walking or jumping motions) are modelled by another subsystem given the computed trajectories. Splitting the mapping into two subsystems leaves the degrees of freedom to the animators for designing their preferred movements. This idea is similar to the framework proposed by [HSK14].

We only consider zero-order information (i.e. positions) based on the advice from practitioners that crowd is typically geometry-based. Depending on the application, if higher-order information such as velocity and acceleration is needed, one may extract and integrate the information into the feature vector. It will require some

extra designs to map multi-order information between gestures and motions in an effective manner.

10. Conclusion and Discussions

In this paper, we propose a data-driven approach for crowd control using a multi-touch device. Our method learns from a set of user-performed gestures and allows a user to intuitively control a crowd. To achieve this, we propose a set of gesture features that represent high-level information of the user-performed gestures. We also propose a method to extract crowd motion features using a mixture of Gaussian processes. Given a run-time gesture, we perform a KNN search in our gesture database, and find the K corresponding crowd motions. We then combine the K crowd motions to control the run-time crowd. Our system runs in real-time and has high control accuracy.

Like many existing systems, the simulation time increases with the number of characters. However, our system is relatively computational efficient with a large number of characters. This is because the majority part of the system is based on the extracted motion features and gesture features, which are independent of the number of characters. The only step that is computationally proportional to the character number is the synthesis of the final crowd motions.

Theoretically speaking, given the right gesture, it is possible to interpolate two classes of crowd motion (e.g. *translate* and *join*) to generate a new run-time motion. However, it rarely happens in practice due to the relatively wide range of gestures we collected to cover the possible variation within the same class. As a result, the interpolation performed is mostly intra-class.

Theoretically, the mapping could be contaminated if the gesture-motion pairs are not generated well, such as two similar gestures generating very different motions or vice versa. In practice, we find that KNN helps to reduce the effect of outlier mappings, as multiple motions/gestures are combined, and less similar ones are given smaller weights. Also, the mapping in our database is very descriptive thanks to the distinctiveness among the types of basic motions, which results in a set of distinctive corresponding gestures. More complex motions can be decomposed into the combinations of basic ones to avoid overcomplicated motion-gesture mappings.

Our system analyzes the gesture in a discrete manner. Each gesture controls the crowd in a short time interval. One possible solution is to apply the continuous recognition algorithms proposed in [SKT11], in which the input gesture is continuously being recognized using a variable size sliding window.

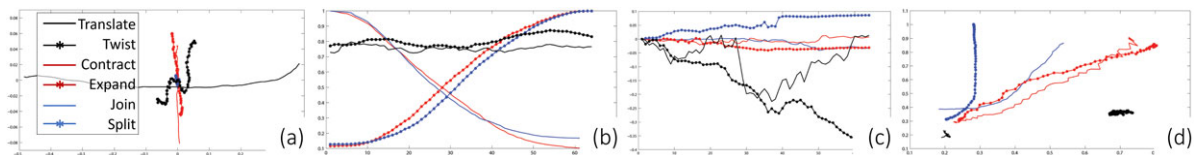


Figure 12: The visualization of gesture features by types: (a) avg. y -position versus avg. x -position for $C(G)$, (b) avg. distance from centroid versus time for $S(G)$, (c) avg. total rotation versus time for $R(G)$, (d) avg. bounding box height versus width for $B(G)$.

An interesting research direction is to introduce more intra-class differences in the crowd motion. For example, we can have a spread-out *translate* crowd motion and a condensed one. We then collect corresponding gesture inputs from the user into the database. As a result, a small gesture difference will generate a small variation of the crowd motion, allowing fine control of the crowd.

Acknowledgements

This project was supported in part by the Engineering and Physical Sciences Research Council (EPSRC) (Ref: EP/M002632/1) and the Royal Society (REF: IE160609).

References

- [BC94] BERNDT D. J., CLIFFORD J.: Using dynamic time warping to find patterns in time series. In *Proceedings of KDD Workshop* (Seattle, WA, USA, 1994), U. M. Fayyad and R. Uthurusamy (Eds.). AAAI Press, pp. 359–370.
- [Bis96] BISHOP C. M.: *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, UK, 1996.
- [BvdPPH11] BONNEEL N., VAN DE PANNE M., PARIS S., HEIDRICH W.: Displacement interpolation using Lagrangian mass transport. In *Proceedings of the 2011 SIGGRAPH Asia Conference SA '11* (Hong Kong, China, 2011).
- [CK08] CHENEY W., KINCAID D. R.: *Linear Algebra: Theory and Applications* (1st edition). Jones and Bartlett Publishers, Inc., USA, 2008.
- [GCG10] GÖRG M. T., CEBULLA M., GARZON S. R.: A framework for abstract representation and recognition of gestures in multi-touch applications. In *Proceedings of Third International Conference on Advances in Computer-Human Interactions, 2010, ACHI '10*. (Saint Maarten, Netherlands Antilles, 2010), IEEE, pp. 143–147.
- [GD11] GU Q., DENG Z.: Formation sketching: An approach to stylize groups in crowd simulation. In *Proceedings of Graphics Interface 2011* (St. John's, Newfoundland, Canada, 2011), Canadian Human-Computer Communications Society, pp. 1–8.
- [GD13] GU Q., DENG Z.: Generating freestyle group formations in agent-based crowd simulations. *IEEE Computer Graphics and Applications* 33, 1 (2013), 20–31.
- [GGH*03] GIBBON D., GUT U., HELL B., LOOKS K., THIES A., TRIPPEL T.: A computational model of arm gestures in conversation. In *Proceedings of INTERSPEECH* (Geneva, Switzerland, 2003).
- [HKS17] HOLDEN D., KOMURA T., SAITO J.: Phase-functioned neural networks for character control. *ACM Transactions on Graphics* 36, 4 (July 2017), 42:1–42:13.
- [HSK12] HENRY J., SHUM H. P. H., KOMURA T.: Environment-aware real-time crowd control. In *EUROSCA'12: Proceedings of the 11th ACM SIGGRAPH/Eurographics Conference on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2012), Eurographics Association, pp. 193–200.
- [HSK14] HENRY J., SHUM H. P. H., KOMURA T.: Interactive formation control in complex environments. *IEEE Transactions on Visualization and Computer Graphics* 20, 2 (2014), 211–222.
- [JCP*10] JU E., CHOI M. G., PARK M., LEE J., LEE K. H., TAKAHASHI S.: Morphable crowds. *ACM Transactions on Graphics* 29, 6 (2010), 140:1–140:10.
- [JTZ*12] JIANG Y., TIAN F., ZHANG X., LIU W., DAI G., WANG H.: Unistroke gestures on multi-touch interaction: Supporting flexible touches with key stroke extraction. In *IUI '12: Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces* (New York, NY, USA, 2012), ACM, pp. 85–88.
- [JXW*08] JIN X., XU J., WANG C. C., HUANG S., ZHANG J.: Interactive control of large-crowd navigation in virtual environments using vector fields. *IEEE Computer Graphics and Applications* 28, 6 (2008), 37–46.
- [KHDA12] KIN K., HARTMANN B., DEROSE T., AGRAWALA M.: Proton++: A customizable declarative multitouch framework. In *UIST '12: Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology* (New York, NY, USA, 2012), ACM, pp. 477–486.
- [KHKL09] KIM M., HYUN K., KIM J., LEE J.: Synchronized multi-character motion editing. In *ACM Transactions on Graphics* 28 (2009), 79:1–79:9.
- [KLLT08] KWON T., LEE K. H., LEE J., TAKAHASHI S.: Group motion editing. *ACM Transactions on Graphics* 27 (2008), 80:1–80:8.
- [KO10] KARAMOOUZAS I., OVERMARS M.: Simulating the local behaviour of small pedestrian groups. In *Proceedings of the 17th ACM Symposium on Virtual Reality Software and Technology* (Hong Kong, 2010), ACM, pp. 183–190.
- [KSII09] KATO J., SAKAMOTO D., INAMI M., IGARASHI T.: Multi-touch interface for controlling multiple mobile robots. In *Proceedings of CHI'09 Extended Abstracts on Human Factors in Computing Systems* (Boston, MA, USA, 2009), ACM, pp. 3443–3448.
- [KSKL14] KIM J., SEOL Y., KWON T., LEE J.: Interactive manipulation of large-scale crowd animation. *ACM Transactions on Graphics* 33, 4 (Jul 2014), 1–10.
- [KWK*10] KAMMER D., WOJDAK J., KECK M., GROH R., TARANKO S.: Towards a formalization of multi-touch gestures. In *ITS '10: Proceedings of ACM International Conference on Interactive Tabletops and Surfaces* (New York, NY, USA, 2010), ACM, pp. 49–58.
- [Lal09] LALESCU C. C.: Two hierarchies of spline interpolations. practical algorithms for multivariate higher order splines. arXiv:0905.3564, 2009.
- [LCHL07] LEE K. H., CHOI M. G., HONG Q., LEE J.: Group behavior from video: A data-driven approach to crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (San Diego, CA, USA, 2007), Eurographics Association, pp. 109–118.

- [LL12] LÜ H., LI Y.: Gesture coder: A tool for programming multi-touch gestures by demonstration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Austin, TX, USA, 2012), ACM, pp. 2875–2884.
- [LL13] LÜ H., LI Y.: Gesture studio: Authoring multi-touch interactions through demonstration and declaration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Paris, France, 2013), ACM, pp. 257–266.
- [LWB*14] LEE Y., WAMPLER K., BERNSTEIN G., POPOVIĆ J., POPOVIĆ Z.: Motion fields for interactive character locomotion. *Communications of the ACM* 57, 6 (Jun2014), 101–108.
- [MC12] MIN J., CHAI J.: Motion graphs++. *ACM Transactions on Graphics* 31, 6 (Nov2012), 153:1–153:12.
- [MCC09] MIN J., CHEN Y.-L., CHAI J.: Interactive generation of human animation with deformable motion models. *ACM Transactions on Graphics* 29, 1 (Dec2009), 1–12.
- [MDC*09] MICIRE M., DESAI M., COURTEMANCHE A., TSUI K. M., YANCO H. A.: Analysis of natural gestures for controlling robot teams on multi-touch tabletop surfaces. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces* (Banff, Alberta, Canada, 2009), ACM, pp. 41–48.
- [OO09] OSHITA M., OGIWARA Y.: Sketch-based interface for crowd animation. In *Proceedings of Smart Graphics* (Salamanca, Spain, 2009), Springer, pp. 253–262.
- [Par10] PARK M. J.: Guiding flows for controlling crowds. *Visual Computer* 26, 11 (2010), 1383–1391.
- [PVDBC*11] PATIL S., VAN DEN BERG J., CURTIS S., LIN M. C., MANOCHA D.: Directing crowd simulations using navigation fields. *IEEE Transactions on Visualization and Computer Graphics* 17, 2 (2011), 244–254.
- [RGR13] REKIK Y., GRISONI L., ROUSSEL N.: Towards Many Gestures to One Command: A User Study for Tabletops. In *Proceedings of INTERACT - 14th IFIP TC13 Conference on Human-Computer Interaction* (Cape Town, South Africa, Sept. 2013), Nelson Mandela Metropolitan University, CSIR Meraka Institute and the University of Cape Town, Springer.
- [RVG14] REKIK Y., VATAVU R.-D., GRISONI L.: Match-up & conquer. In *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces - AVI '14* (Como, Italy, 2014), pp. 201–208.
- [SCOL*04] SORKINE O., COHEN-OR D., LIPMAN Y., ALEXA M., RÖSSL C., SEIDEL H.-P.: Laplacian surface editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry Processing* (Nice, France, 2004), ACM, pp. 175–184.
- [SH12] SHUM H. P. H., HO E. S. L.: Real-time physical modelling of character movements with microsoft kinect. In *VRST '12: Proceedings of the 18th ACM Symposium on Virtual Reality Software and Technology* (New York, NY, USA, Dec 2012), ACM, pp. 17–24.
- [SKT11] SHUM H. P. H., KOMURA T., TAKAGI S.: Fast accelerometer-based motion recognition with a dual buffer framework. *International Journal of Virtual Reality* 10, 3 (Sep2011), 17–24.
- [SKY12] SHUM H. P. H., KOMURA T., YAMAZAKI S.: Simulating multiple character interactions with collaborative and adversarial goals. *IEEE Transactions on Visualization and Computer Graphics* 18, 5 (May2012), 741–752.
- [SMTT*17] STUVEL S. A., MAGNENAT-THALMANN N., THALMANN D., VAN DER STAPPEN A. F., EGGS A., undefined, undefined, undefined, undefined: Torso crowds. *IEEE Transactions on Visualization and Computer Graphics* 23, 7 (2017), 1823–1837.
- [TCP06] TREUILLE A., COOPER S., POPOVIĆ Z.: Continuum crowds. In *ACM Transactions on Graphics* 25 (2006), 1160–1168.
- [Tou83] TOUSSAINT G.: Solving geometric problems with the rotating calipers. In *Proc. IEEE Melecon* (1983).
- [TYK*09] TAKAHASHI S., YOSHIDA K., KWON T., LEE K. H., LEE J., SHIN S. Y.: Spectral-based group formation control. *Computer Graphics Forum* 28 (2009), 639–648.
- [VAW12] VATAVU R.-D., ANTHONY L., WOBBEROCK J. O.: Gestures as point clouds: A \$P recognizer for user interface prototypes. In *ICMI '12: Proceedings of the 14th ACM International Conference on Multimodal Interaction* (New York, NY, USA, 2012), ACM, pp. 273–280.
- [vdBLM08] VAN DEN BERG J., LIN M., MANOCHA D.: Reciprocal velocity obstacles for real-time multi-agent navigation. In *Proceedings of 2008 IEEE International Conference on Robotics and Automation* (Pasadena, CA, USA, May 2008), pp. 1928–1935.
- [WOO16] WANG H., ONDŘEJ J., O'SULLIVAN C.: Path patterns: Analyzing and comparing real and simulated crowds. In *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games 2016* (Redmond, WA, USA, 2016), ACM, pp. 49–57.
- [WWL07] WOBBEROCK J. O., WILSON A. D., LI Y.: Gestures without libraries, toolkits or training: A \$I recognizer for user interface prototypes. In *UIST '07: Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology* (New York, NY, USA, 2007), ACM, pp. 159–168.
- [XWY*15] XU M., WU Y., YE Y., FARKAS I., JIANG H., DENG Z.: Collective crowd formation transform with mutual information-based runtime feedback. *Computer Graphics Forum* 34 (2015), 60–73.
- [ZZC*14] ZHENG L., ZHAO J., CHENG Y., CHEN H., LIU X., WANG W.: Geometry-constrained crowd formation animation. *Computers & Graphics* 38 (2014), 268–276.

Supporting Information

Additional Supporting Information may be found in the online version of this article at the publisher's web site:

Video S1