Learning Motion Rules for Autonomous Characters from Control Logs Using Support Vector Machine

Masaki Oshita
Kyushu Institute of Technology
oshita@ces.kyutech.ac.jp

Tomomasa Yoshiya
Kyushu Institute of Technology
yoshiya@cg.ces.kyutech.ac.jp

Abstract

In this paper, we propose a method for automatically learning motion rules for autonomous characters from control logs. We ask a few players to control their characters in a gaming application. Based on the control logs obtained, motion rules are constructed for each character. We use a support vector machine (SVM) to learn the motion rules. However, the SVM cannot be applied directly to our problem, as the necessary variables and their values vary according to the motion. To solve this problem, we introduce a layered mechanism and adaptive parameters to the SVM. As an experiment, our method has been applied in simulating football players.

Keywords: motion rules, autonomous characters, support vector machine

1. Introduction

Recently, autonomous characters have been used in many applications, including computer games and computer animation. Non-player characters (enemies, allies, etc.) in computer games are expected to act naturally when responding to user-controlled characters, other autonomous characters, and the environment. In addition, crowd scenes in movies can be created efficiently by arranging many autonomous characters on the virtual scene and letting them interact with one another. To control such autonomous characters, their individual motion rules must be known. A motion rule defines the condition under that the character performs a motion. For example, the motion rules for a soldier character may include attacking an enemy when it is in a certain range, escaping when attacked, etc.

Defining motion rules is a very difficult and time consuming task. A designer needs to describe many rules and their parameters, which are tuned through trial and error. Moreover, since human behavior is sometimes based on instinct, which does not conform to obvious rules, it is hard to describe such rules.

In this paper, we propose a method for automatically learning motion rules for autonomous characters from control logs. First, we ask a few players to control their characters in a gaming application. Based on the control logs created, motion rules for each character are constructed. We assume that a character's motion is dependent on various conditions including the character's state, other characters' states, and the scene. Sometimes an autonomous character is expected to react to several other characters in a crowd scene. Even in this case, however, the character's action is deemed to be determined by the few characters close by. For example, a soldier's motion can be determined from a few other important characters such as his commander, the opponent he is trying to attack, and the opponent trying to attack him. Thus, motion rules learned from a few human players should be adaptable to characters in a crowd scene.

We use a support vector machine (SVM) [1][2] to learn motion rules. However, the SVM cannot be applied directly to our problem, as the necessary variables and their values vary according to the motion. To solve this problem, we have introduced a layered mechanism and adaptive parameters to the SVM. As an experiment, our method has been applied in simulating football players.

2. Related Work

Recently, an animation system specific to crowd animation, MASSIVE [3], has been widely used in movie production. MASSIVE provides a sophisticated GUI to describe motion rules using fuzzy rules. However, it is still difficult to design effective motion rules.
There are methods for acquiring parameters automatically for crowd simulation. Lerner et al. [4] estimated the moving velocity and direction at each point in the scene from videos of real pedestrians taken from above.
Hoshino et al. [5] used a similar method to ours for learning motion rules for fighting game characters from control logs. However, they simply choose a sequence of actions based on the amount of successful training data.
Some researchers have employed SVMs in the computer animation field. Faloutsos et al. [6] used one for selecting controllers for physics simulations based on a character's condition, while Zordan et al. [7] used one for selecting the anticipating motion for an incoming impact.

3. System Overview

The data flow in the proposed approach is shown in Figure 1. In order to learn the motion rules, control logs are acquired from human players using a gaming application. Usually, when a designer wishes to create motion rules for autonomous characters in a computer game, control logs can be acquired by having human players control the autonomous characters in the application.
The system then extracts training data from the control logs for learning motion rules. The training data contain sets of an executed motion type (e.g., move, attack, defend) and a feature vector representing the state in which the motion was executed. The feature vector contains variables that are relevant to the motion rules (e.g., self position, opponents' positions, target position). The action types and feature vector depend on the application and character.
Motion rules represent the conditions of the feature vector in executing each motion and are learned from the training data. By using the motion rules (SVM), the system can determine which motion should be executed based on a particular feature vector.

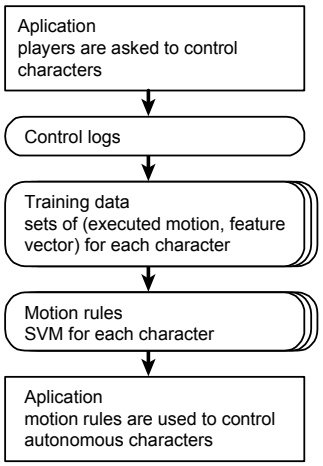

Figure 1 : Data flow in the proposed approach

At runtime, based on the feature vector computed from the current state of the scene, the character's motion is determined from the motion rules (SVM). In our system, when a motion (e.g., one walking motion cycle, an attacking motion) terminates, the character's next motion is determined.
The character's animation is realized by using a conventional method. Our system employs a motion tree, which is popular in computer games. A motion tree contains short motion data and possible transitions between these. By playing back the motion data while sequentially traversing the motion tree, we can generate continuous animation for a character. Each motion type is assigned to a corresponding short motion. By selecting a motion type from a user's control or motion rules, animation for a character is generated.
Each character can have a different motion tree and motion rules or share these with other characters. If more than one character behaves in the same way (e.g., soldiers), they can share a motion tree and motion rules. If a user needs to acquire motion rules for more than one type of character simultaneously, an equivalent number of players must play together.

4. Motion Rule Representation

4.1 Support Vector Machine

As explained earlier, we use a support vector machine (SVM) [1] to learn motion rules. The SVM is a popular pattern recognition technique. When there are training data in a feature space and each item of the training data belongs to one of two classes, the SVM calculates a

hyperplane that minimizes the distance between the hyperplane and the closest training data for each class. The SVM is extended to multiple classes and non-linear hyperplanes by mapping the feature space to a latent higher dimensional space. We use a LIBSVM [2] for our implementation.
Alternative recognition techniques, such as neural networks, Hidden Markov Models, k-nearest neighbours, also exist. Compared to these, the SVM is considered to be robust for high dimensional and unknown data.

4.2 Problems of SVM

There are, however, two major problems in applying an SVM to our application.
First, a large number of training data are required when using a high dimensional feature space. Even if only a few of the variables in the feature space are important for a motion, many training data are necessary to learn that. Without sufficient training data, classification becomes unstable and an unexpected motion may be selected.
Second, it is difficult to tune the size of classification. The LIBSVM uses a Radial Basis Function (RBF) Kernel to partition the feature space and the radius of RBF, γ, must be specified by the user. However, the appropriate parameter depends on the regions. If the γ parameter (radius) is too large, it cannot classify a region containing a large number of training data of different classes. On the other hand, if the γ parameter is too small, a large region containing a small number of training data will be divided into smaller regions belonging to the wrong classes.
These problems can be avoided if there are no regions too small and if many training data are available especially when these can be generated automatically using simulation [6][7]. In our application, however, subtle conditions (i.e., small regions in the feature space) may exist for some motions and training data must be acquired from humans.

4.3 Layered SVM

To solve the first problem, we introduce the layered SVM consisting of several SVM models that are constructed from different subsets of the feature vector, in addition to the SVM model created from the full set of the feature vector. At runtime, the system chooses the appropriate SVM model. First, we divide

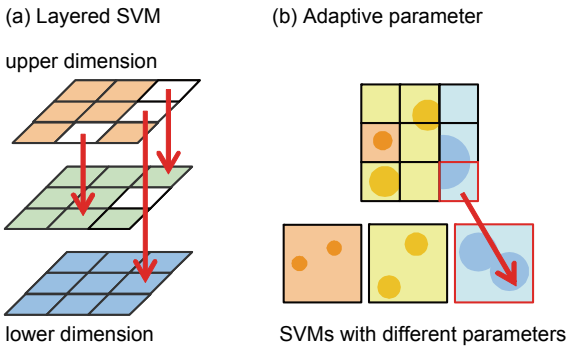

Figure 2: Two extensions to the SVM

the feature space into a grid. Then, the appropriate level of SVM model is assigned to each cell of the grid, based on whether training data exist in the SVM model (Figure 2(a)). If there is not enough data at a higher level of the grid, the system uses a lower level of the SVM. If there is not enough data for any SVM model, it uses the lowest level of the SVM, because it is expected to be more reliable in this case.

4.4 Adaptive Parameter

To solve the second problem, we introduce adaptive parameters. As explained in the previous subsection, each SVM in the layered SVM is divided into a grid. We assign a different parameter to each cell of the grid.
To realize this, we construct several SVMs using different parameters for each layer. Then, for each grid cell, we determine which SVM should be used (Figure 2(b)). The system calculates the recognition rate for each SVM using the training data at the cell and then chooses the best SVM for the cell.

5. Experimental Results

We have tested the proposed method on a football gaming application in which three characters, that is, offence, defence, and goal keeper, are controlled using gamepads. Each character can move in eight directions (Figure 3(a)). In addition, the offence can shoot forwards towards the left, right, or center of the goal. If the defence or goal keeper touches the ball, he keeps the ball. The game ends when the offence scores or the opponents get the ball. We asked three subjects to play the game together and used their control logs to learn the motion rules for each character. In our experiments, we used the control logs of 20 games (about two minutes). It took about 5

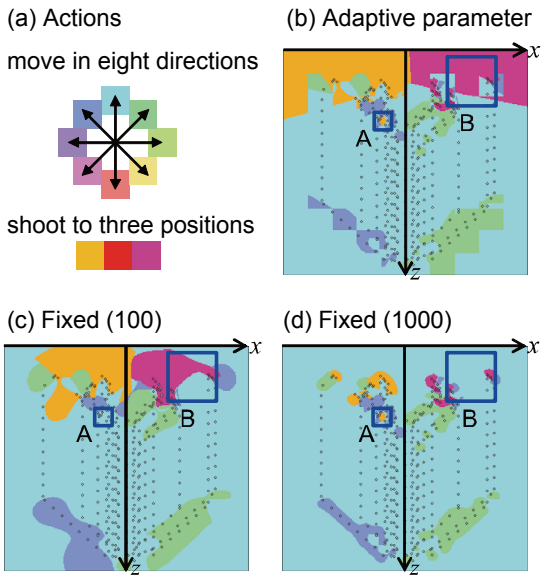

Figure 3: Results of adaptive parameter

minutes to learn the motion rules on a Pentium 4, 3.2GHz CPU. At runtime, animations are generated in real-time. The generated animations, included on the accompanying video, show all three characters being controlled autonomously using the learned motion rules.

5.1 Evaluation of Adaptive Parameter

First, we show the evaluation results for the effectiveness of the adaptive parameters. Figure 3 shows the learned SVM models for the offence using adaptive and fixed parameters (γ = 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000; although only 100 and 1000 are shown in the figure) in a two-dimensional feature space (offence's position).

With a small parameter (γ =100, Figure 3(c)), small regions are missing (see area A), whereas with a large parameter (γ =1000, Figure 3(d)) large regions are missing (see area B). Using our adaptive parameters (Figure 3(b)), appropriate parameters are chosen for each grid and both small and large regions are preserved.

5.2 Evaluation of Layered SVM

Next, we show the evaluation results for the effectiveness of the layered SVM. As shown in Table 1, we construct four layers of SVM models with different feature vectors.

The generated animations are shown on the accompanying video. With a higher dimensional model (Model 1), even when the offence gets close to the goal and is expected to shoot, this sometimes does not happen

Table 1: Layered SVM models for the offence

No.	Feature vector	Dim.
1	offence, defence, keeper positions	6
2	offence, defence positions	4
3	offence, keeper positions	4
4	offence position	2

because no training data exist for a shoot motion matching the defence position. With the layered SVM, in this case, the defence position is disregarded and a shoot motion is executed adaptively using the lower dimensional model (Model 3). If the defence position needs to be considered (e.g., when heading a goal while avoiding the defence), a higher dimensional model is used.

6. Conclusion

In this paper, we proposed a method for constructing motion rules from control logs. By introducing a layered mechanism and adaptive parameters to the SVM, we realized plausible motion from small control logs. Future work includes determining how many control logs are necessary and applying our method to real applications with higher dimensions. In addition, instead of using control logs, by analyzing motion capture data of several performers, we can learn motion rules and construct a motion graph [8] at the same time. Pursuing this alternative approach is included in our future work.

References

[1] V. N. Vapnik. The Nature of Statistical Learning Theory, Springer, 2000.

[2] C. C. Chang, C. J. Lin. LIBSVM. http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[3] MASSIVE. http://www.massivesoftware.com/

[4] A. Lerner, Y. Chrysanthou, D. Lischi. Crowds by example. Computer Graphics Forum (Eurographics 2007), vol. 26, no. 3, pp. 655-664, 2007.

[5] J. Hoshino, A. Tanaka, K. Hamana. The Fighting Game Character that Grows up by Imitation Learning. IPSJ Journal, vol. 49, no. 7, pp. 2539-2548, 2008.

[6] P. Faloutsos, M. van de Panne, D. Terzopoulos. Composable Controllers for Character Animation. SIGGRAPH 2001, pp. 251-260, 2001.

[7] V. Zordan et al. Anticipation from Example. ACM Virtual Reality Software and Technology 2007, pp. 81-84, 2007.

[8] K. Kovar, M. Gleicher, F. Pighin. Motion Graphs. ACM Transactions on Graphics (SIGGRAPH 2002), vol. 21, issue 3, pp. 473-482, 2002