Fast Accelerometer-Based Motion Recognition with a Dual Buffer Framework

Hubert P. H. Shum, Taku Komura and Shu Takagi

***Abstract*—The low-cost gyro-accelerometer based controllers have opened up a potential of using 3D computer games for serious applications such as sports and vocational training. Previous games apply simple template matching for the motion recognition which suffers from poor accuracy and time lag. In order to cope with these problems, we propose a novel dual buffer approach that dramatically increases the recognition rate and shortens the time lag for motion recognition. The system first recognizes the user control signal with a small buffer to minimize the time lag. When more signals arrive from the sensor, an elaborate recognition is performed, and the previously recognized action is switched if necessary. Using boxing as an example, we show that we can control a virtual character to perform 13 different actions using a buffer size of one-tenth of a second. Since our system is computationally inexpensive, it can be used in game consoles. As it is accurate and responsive, it can also be applied for serious sport training.**

I. Introduction

Inertia-based sensors are currently the most low-cost and widely used sensors for recognizing the human movements in real-time. For controlling virtual characters in computer games, the system analyzes the acceleration profile of the controller and matches them with the template profiles, and finally the corresponding action is launched by the virtual character.

A simple template matching algorithm could work satisfactory for simple action games, in which the number of available actions is limited and the response time is less important. However, it is not good enough for applications such as sports training, in which the system needs to recognize a wide variety of actions in a very short time. Furthermore, even for entertainment, due to the increasing expectation of the game quality, we need better algorithms for controlling virtual characters.

In order to cope with this problem, we propose a novel dual buffer approach that dramatically increases the recognition rate and shortens the time lag for motion recognition. We first recognize the motion using a short sequence of signals to ensure fast system response. As more control signals arrive, an elaborate recognition is performed using the longer sequence of signals. If the action recognized is different from the results of the previous recognition, the virtual character will switch the performing action to the newly matched one. Using boxing as an example, we show that we can control a virtual character to perform 13 different actions using a buffer size of one-tenth of a second.

To demonstrate the effectiveness of our approach, we have implemented our algorithm on a boxing training system in which the user can interactively play boxing with the virtual opponent. The overview of our system is shown in Figure 1. We capture the movements of a boxer, and segment them into semantic segments called actions. The actions are organized into a structure called action level motion graph [1] (Figure 1 middle left). We generate an Interaction Graph [2] which is used to control characters such that they intelligently interact with the others in real-time (Figure 1 left). For the user controlled character, we select some actions as controllable actions, and associate each of them with a control signal from the accelerometer (Figure 1 middle right). During run-time, we collect signals from the accelerometers, and perform matching to analyze which action the user is performing (Figure 1 right).

The paper is organized as follows: the related works are described in Section II. The precomputation process to create the actions and control signals database is described in Section III. The dual buffer approach for fast action recognition is explained in Section IV. How we animate the actions of the character with minimal artifacts is explained in Section V. We test and evaluate our system in Section VI. We discuss about our method in Section VII. Finally, we conclude our paper and state the possible future directions in Section VIII.

II. Related Work

In this research, we create an action database and use accelerometers to control the virtual characters. Therefore, we first review methods to control characters using low-degrees of freedom controllers. During the experiment, we create a boxing game to test our system. The opponent is an intelligent computer-controlled character trained by sampling-based reinforcement learning. Therefore, we next discuss researches related to machine learning.

A. Character Control by Low Degrees of Freedom Sensors

In computer games, the user needs to control virtual characters using a device with low degrees of freedom, such as joysticks or accelerometers-based controllers like the Wiimote. Chai and Hodgins [3] proposed a data-driven approach to control human characters with high degrees of freedom using the control signals of low degrees of freedom. Shiratori et al. [4] propose a method to control walking character by attaching a Wii controller at each leg, and calculating the frequency of

Hubert P. H. Shum is with the University of Worcester. E-Mail: h.shum@worc.ac.uk

Taku Komura is with the University of Edinburgh. E-Mail: tkomura@inf.ed.ac.uk

Shu Takagi is with RIKEN, Japan. E-Mail: takagish@riken.jp

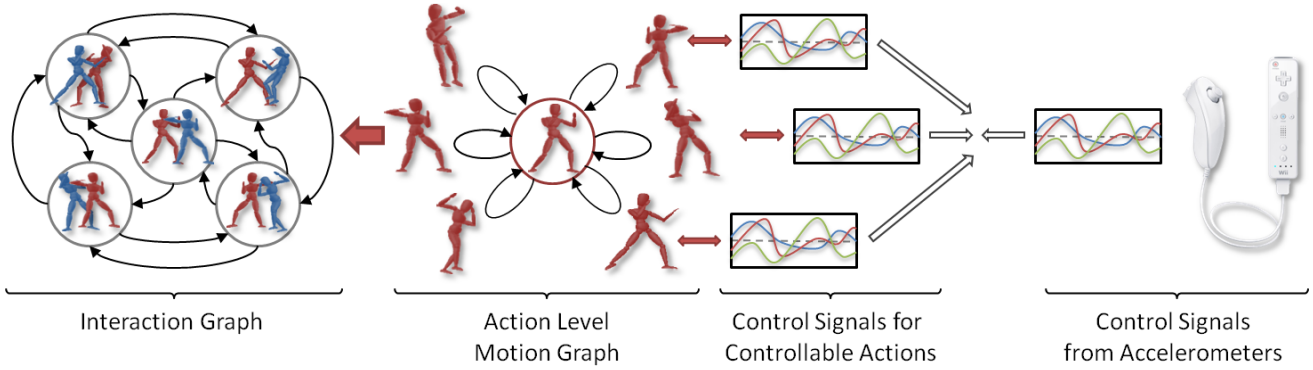


Fig. 1. The overview of our system. An action level motion graph is created based on captured motion. Some actions are selected to be controllable actions. Each of them is associated with a control signal profile. During run-time, we observe the control signals from the accelerometers and perform action matching. On the other hand, we create an Interaction Graph based on the motion graph to control the computer players.

the stepping motion. This kind of approach cannot be applied to control characters with a large number of actions.

In the game industry, a lot of methods to control virtual characters using the Wiimote accelerometers have been invented. In the game "The Legend of Zelda: Twilight Princess", the user can let the character swing the sword by shaking the Wiimote. In "Mario Kart Wii", the user can tilt the controller to specify the driving direction of a car. Although such control methods are applicable for system that only needs to specify the orientation or the amplitude of some movements, they cannot be used for recognition of complex full body motions in sports.

For some games such as "Wii Sports", the system can classify the user control signals into action groups. For example, in the boxing game, users can launch different punching movements such as upper cuts and straight punches by changing the direction of moving the controller. The major problem is the trade-off between recognition accuracy and time lag. We focus on this problem in this paper, and will discuss more in later sections.

B. Character Control by Machine Learning

Automatically learning to control characters intelligently is a topic attracting many researchers due to its applicability to interactive systems such as computer games. Treuille et al. [5] trains characters to avoid obstacles and other characters when walking in the virtual environment. Lee and Lee [6] train the characters to efficiently approach and hit the target, which is directly applicable for boxing games. McCann and Pollard [7] applies reinforcement learning to let the system learn how the user tends to control the character at various situations and let the character prepare for such actions in advance. These kinds of training are rather easy as the state space is relatively small.

Machine learning can also be used for training the computer-controlled character to intelligently interact with a user controlled character. Graepel et al. [8] trained the computer-controlled character in fighting games to intelligently compete with user controlled characters. The problem when handling cases where multiple characters interact with each other is that the state space becomes too large to be fully explored. Shum et al. [2] defined several general criteria to evaluate the importance of the states, and composed a finite state machine by collecting samples of the important states. In this research, we will apply the same approach and make use of the precomputed finite state machine to limit the choices of actions that can be launched by the user-controlled character.

III. Database for Actions and Control Signals

In this section, we explain the process to create our action database for generating computer controlled characters. We also explain how to associate the control signals to the actions that are usable by the user controlled characters.

We capture the required motion and organize them into a data structure called the motion graph. Long sequences of shadow boxing motion from a professional boxer are captured using an optical motion capture system. The motion data is segmented into semantic segments called actions automatically. For boxing, an action can be a straight punch, a step forward, a defense, or any combination of them. These actions are tagged based on their nature and used to create an action level motion graph, in which a node represents a posture and an edge represents an action. The readers are referred to [1] for further details.

In our system, the action level motion graph is simplified for easier control (Figure 1 middle left). The graph is basically a fat graph [9] with actions as entities. There may be many fat nodes, which are standard postures for starting and ending most of the actions. To simplify the control system, we extract the biggest fat node and the edges connecting to it, and discard all edges connecting to other nodes. We select some actions as controllable actions that can be controlled by the human player. Since the graph consists of one fat node only, we can always be sure that the user can launch any controllable action when the character is in the standard posture.

We create a finite state machine called the Interaction Graph to control the computer players (Figure 1 left). Based on the action level motion graph, we can generate states of interactions for two characters. We collect the states considering the quality of the interactions and how often they are visited. With the collected states, we create the Interaction Graph in which the nodes are the states of interaction, while the edges are actions performed by one of the characters. By applying

dynamics programming, we can precompute the optimal actions to be performed for any states of interaction. Therefore, with the Interaction Graph, we can control intelligent computer players in real-time with minimal computational cost. The readers are referred to [2] for further details.

We associate some specific actions with the control signals of the input device for the real-time control of the user-controlled character (Figure 1 middle right). Each control signal consists of a stream of 6 dimensional signals representing the 3 dimensional acceleration of the two hands at each frame. Since the user may not perform the action steady and accurately, we capture multiple trials of the control signal for the same action. The samples are aligned by dynamic time warping, and the mean value at each frame is calculated:

$$E(s(t)) = \sum_{i=0}^{n_{sample}} (s_i(t))/n_{sample} \quad (1)$$

where $s_i(t)$ is the 6D signal at frame t of the i^{th} sample, n_{sample} is the total number of samples. The calculated control signal is then associated to the action for run-time matching.

During run-time, we collect signals from the accelerometer and match them with the signals stored in the database. The process will be explained in the next section.

IV. Run-Time Recognition with Dual Buffer

In the section, we explain our framework of using dual buffer to achieve fast action recognition. We will explain the process of buffering control signals, recognizing the actions, and synchronizing the user action with rendering.

When matching actions, a number of signals have to be cumulated before the recognition starts. This buffer is needed to increase the accuracy of action matching. An optimal matching can be performed if the buffer is larger than the action duration. Traditional approaches use a signal buffer to store the control signals. If the required buffer size is large, there will be a lag between the time when the user starts performing an action and the time when the character starts acting. However, with a small buffer, the recognition is inaccurate due to the lack of control information. To cope with this dilemma, we propose a dual buffer approach which consists of a main buffer and a supplementary buffer. The main buffer is a small buffer to conduct a basic, rough action matching. The supplementary buffer starts to store the signals when the main buffer is full, and provides extra information for further matching.

A. Control Signals Management

In this section, we explain how we cumulate the control signals with the dual buffer.

For every frame, we receive a control signal from the accelerometer. This is visualized in Figure 2(a), in which each green block represents the control signal for one frame, the red rectangle is the main buffer, and the blue rectangle is the supplementary buffer. The signal will be considered as noise and discarded if the sum of the magnitude from all accelerometer is smaller than a predefined threshold, which is set to 1.0G in our system. Once a valid signal is observed, we push the signal, as well as any forthcoming signals even if they contain no acceleration, into the main buffer.

When the main buffer is full, we start the action recognition thread which runs in background (Figure 2(b)). The oldest signal from the main buffer is shifted to the supplementary buffer, and we store the new signal at the beginning of the main buffer (Figure 2c). When comparing the signals with those of the action data, the signals in both the main and supplementary buffer are used for matching.

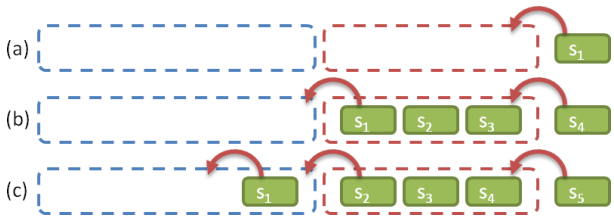


Fig. 2. (a) The main buffer is cumulating control signal from the accelerometers. (b) The recognition thread starts when the main buffer is full. (c) After filling up the main buffer, signals are moved from the main buffer to the supplementary buffer.

If the supplementary buffer is full, the oldest signal will be discarded. However, as we set the supplementary buffer size large, we usually reach a buffer clear condition, which is explained in Section IV-C, before the buffer gets full.

B. Action Recognition

In this section, we explain the action recognition thread that matches the current signal with those in the database. We also explain the two high level commands issued by the action recognition thread once it completes a loop.

We implement a separate action recognition thread such that the process would not affect the rendering speed when using a multi-core CPU (Figure 3). This thread is launched when the main buffer is full. It repeatedly analyzes the user control signal and provides command to the user controlled character. In each loop, it concatenates the signals in the main and supplementary buffer to form a control signal, and applies dynamic programming for finding the most similar signal in the database. At the first loop, since the supplementary buffer is empty, matching is not always accurate. When the supplementary buffer is gradually filled, a better matching is performed and the character switches to a new action if the previous matching is found to be incorrect.

We apply dynamic time warping (DTW) [10] to find the minimum difference between the observed signal and the signals stored in the action database. Let us assume the signal performed by the user to be $s_u(t_u)$, where $t_u \in [0, t_{u_{total}}]$ is a frame number, and a signal in the database to be $s_d(t_d)$, where $t_d \in [0, t_{d_{total}}]$. We first map the initial frame of the observed signal $(s_u(0))$ to that of the signal in the database $(s_d(0))$. We apply DTW to every signal in the database when matching it with the control signal. The one with the smallest difference is selected as the best matching action.

Upon finishing each loop, the thread gives two high level commands. If the currently-played signal is still the best

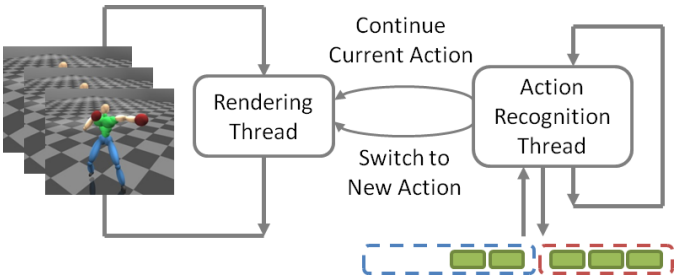


Fig. 3. The action recognition thread runs separately from the rendering thread. It reads both buffer and recognizes the user performed action. Upon finishing one loop, it gives one of the two high level commands to control the character.

matching one, the character continues the current action. If the newly matched signal is a better match, the currently-played action is switched to the newly matched signal. Notice that since some frames (let us say, the first n_p frames) of the previous action have already been played, there will be significant artifacts if we start to play the newly matched action from the beginning. Therefore, we skip the initial part and start playing from the n_p-th frame. Using this approach, the noticeable artifacts can be minimized and a smooth transition is produced in most cases.

Figure 4 shows the effect of our switching method. We use an example in which the hook punch performed by the user is wrongly recognized as a straight punch in the beginning and is amended to the correct hook punch in the middle. The hand trajectories of a straight and hook punch in the motion database are shown in Figure 4 top-left and top-right, respectively. The blue dots represent the positions of the left hand of a straight punch, while the red dots represent that of a hook punch. Figure 4 lower-left shows that our system smoothly switches the straight punch to the hook punch by skipping the initial part. Figure 4 lower-right shows the result if we play the hook punch from the beginning. The transition becomes very unnatural.

C. Synchronization between the Virtual Character and the Human Player

In this section, we explain about the operations needed due to the desynchronization of the action by the user and the game character. As the duration of the user's movement is different from that of the action in the database, either's movement will finish earlier. This desynchronization can cause various problems such as wrongly recognizing the following movement by the user. We first explain the operation when the user's action ends earlier than the character's action. Next, we explain the operation when the character's action ends earlier than the user's action.

The user's action ends earlier when the user withdraws the performing action, or the user performs the action faster than the action in the database. This can lead to the system wrongly matching the user's next action with the ending part of the character's current action. Fortunately, we observed that when a human performs two actions subsequently, usually there is a stabilizing period in between. We call this stabilizing period

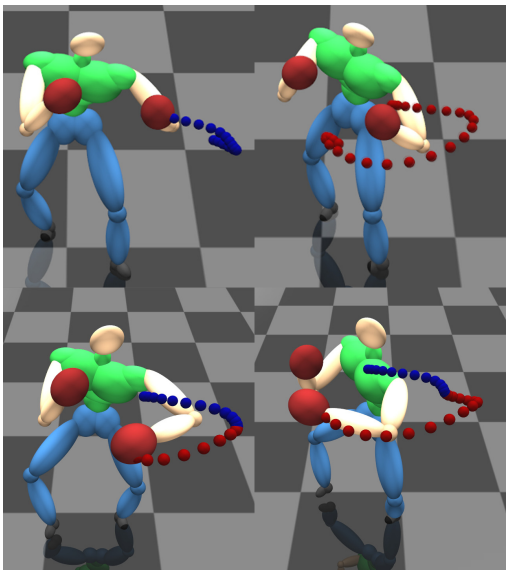


Fig. 4. Action switching when the first action is concluded to be wrongly recognized. The original actions are shown at the top. The resultant movement is smooth if we skip some frames at the second action (lower left). Without frame skipping, there will be significant artifacts (lower right)

the null signal, in which the magnitudes of the accelerometers are under a threshold for β continuous frames. When a null signal is observed, we assume the action is finished and terminate the action currently performed by the virtual character. Notice that there may be artifacts when a large part of the action is not rendered. However, this rarely happens unless the user withdraws the current action in the middle. In our system, we set β to 6, which is equivalent to 1/10 second. Figure 5a shows an example where the user's movement finishes earlier than the character's action. Since we detect a number of signals with no acceleration, which are represented by the white boxes s_6 to s_8, we conclude that the user action has ended and terminates the character's action.

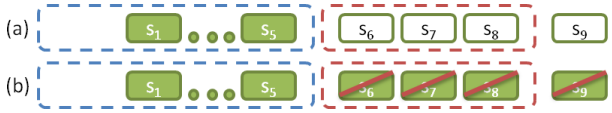


Fig. 5. (a) The user action ends earlier than the rendering action. (b) The rendering action ends earlier than the user action.

The second case of desynchronization happens when the character's action ends earlier than the user's action. This can lead to the system wrongly recognizing the ending part of the user's action as a new action. To avoid this situation, whenever the rendering action is finished while the user is performing something, we discard any forthcoming signal until a null signal is observed, or the number of frame discarded reaches a predefined number, which is set to 20 frames (1/3 second) in our system. Figure 5b shows an example where the character' action finishes earlier than the user's action. When the accelerometer produces signal s_9, the character's action is finished. We discard the forthcoming signals and signal s_6 to s_9 are not passed to the recognition thread. In terms of animation, we keep the character keep standing until the

user starts the next movement and a new matching result is generated.

D. Effect of Buffer Size and Sampling Frequency

In this section, we discuss about the appropriate buffer size and the sampling frequency of the control signals.

The main buffer size affects the time lag between the user and character actions, and the accuracy of the initial action recognition. If the main buffer size is small, the virtual character starts to act immediately after the user starts moving. However, since the first recognition loop takes into account only a small buffer of signals, the accuracy of the recognition is low. On the other hand, if the buffer size is large, the time lag between the user and character actions will be large. In general, appropriate buffer size must be adjusted according to the actions in the database. If the controllable actions in the database are very different from each other in the beginning, we can use a small main buffer. Otherwise, we will need a larger buffer. In our boxing system, the main buffer size is set to 6 frames (1/10 second).

The supplementary buffer size has minimal effect to the system, unless it is unreasonably small. The major usage of the supplementary buffer is to improve the accuracy of recognition when more control signals are available, and it will be cleared once the action is finished. Therefore, its size should be set longer than the longest controllable action in the database. In our system, we set it to be 60 frames (1 second), which is twice the size of the longest action.

We can adjust the sampling frequency to balance the computational power and accuracy of action recognition. If we increase the sampling rate of the control signals, we can detect smaller changes of the signals, and thus the matching will be more accurate. However, it will also increase the computational load of doing the DTW. On the other hand, if the sampling rate is decreased, matching is fast but we lose the details of the control signals. In general, the minimal sampling frequency depends on the accuracy of the accelerometers. With good quality accelerometers, we can rely on each frame of the control signal, and thus do not require high sampling frequency. In our system, we set the frequency to be 60Hz.

V. Character Animation

The characters and the virtual world in our system are created with the Open Dynamics Engine (ODE) [11]. The body parts are modeled by capsules and blocks to reduce the computational cost of collision detection. Whenever a collision is detected, ODE applies repulsive force to the colliding parts to avoid body penetration.

We implement Jakobsen's [12] technique in the ODE to simulate the rigid body dynamics. Each joint is considered as a particle in a particle system, while the body parts are linkages between the particles. In each frame, a reference posture is obtained based on the captured action. The joints are guided to the posture by applying external force generated by a PD control [13]. This is particularly important during action switching as described in Section 7, in which the reference posture may not be continuous. Since we are using external force to control the movements of the character, we will always obtain smooth transitions from discontinuous reference postures. The readers are referred to [14] for more details.

VI. Experiments

In this section, we discuss our experiments based on a serious boxing game.

A. Creating the Motion Database and the Interaction Graph

We captured the shadow boxing motions of a professional boxer for around 7 minutes. The motions were segmented into 148 semantic actions, which were used to create an action level motion graph. Based on the motion graph, we sampled states of interaction and created an Interaction Graph. The graph consists of 55560 nodes and 107167 edges, and was used to control the computer players.

B. Control System

Next, we associated each character action to the control signals and tuned the buffer size of the control signals.

We selected 13 actions from the action level motion graph as controllable actions. They include 2 straight punches, 2 upper cuts, 2 hook punches, 2 parries, 1 blocking and 4 movement actions. We used the Wiimote as our accelerometer since they are available at a reasonable price. For each controllable action, we asked the player to mimic the action 3 times and capture the signal with the Wiimote. We then generated a control signal, which was associated to the controllable action, by aligning and averaging the signal samples captured.

During run-time, both the animation frame-rate and action recognition rate were set to 60Hz. Our system can run in real-time with a notebook computer with a Core 2 Duo P8600 (2.4GHz) CPU and 4GB of RAM. The main buffer size is set to 6 frames (1/10 second) and the supplementary buffer size is set to 60 frames (1 second). More analysis on buffer size can be found at Section VI-D.

The response time of the virtual character is a sum of the main buffer and the time required for recognition. However, the latter is always smaller than $5ms$ in our system, and hence is negligible. Notice that the recognition time increases linearly with respect to the number of controllable actions. If the number is large, we will need to consider the recognition time as part of the time lag.

C. Boxing Game

Here, we describe the experimental results when we applied our algorithm to a boxing game.

In the first experiment, we tested the controllability of the user-controlled character (Figure 6). We asked the player to perform the controllable actions randomly. We found that our system can accurately recognize the user's action with very short response. The lag was considerably shorter than most commercial games nowadays. The accuracy was around 95%. By careful examination, we also found that from time to time the character switches to a correct action because of the

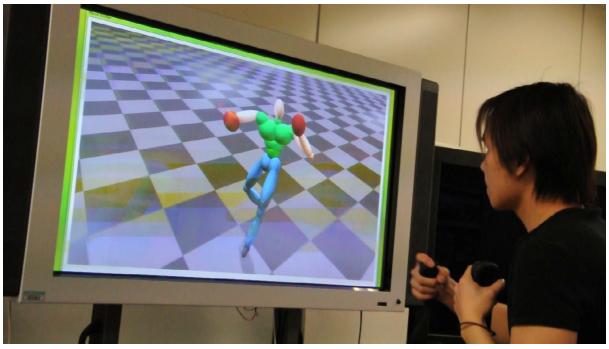


Fig. 6. The user controls a virtual character with a Wiimote. We can achieve a high accuracy with a short control time lag based on our dual buffer framework.

supplementary buffer, but such kind of switching is usually unnoticeable during normal game-play.

In the second experiment, we conducted a virtual boxing match between the user and the computer controlled character (Figure 7). The user controls the virtual character through Wiimote, and the Interaction Graph is used to control the computer-controlled character. When a character is hit, we stop the current action and insert an appropriate falling back action based on the attack speed and direction [15]. For the user-controlled character, any control signals from the user were discarded during the falling back action.

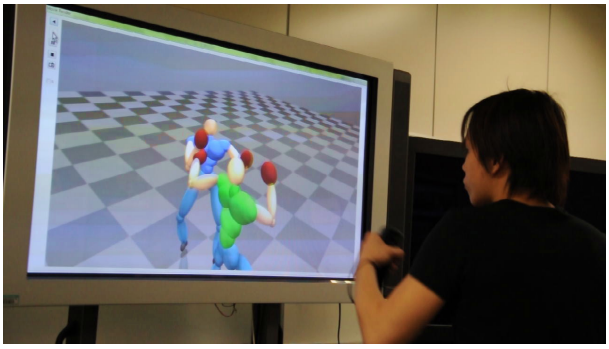


Fig. 7. The user boxes with a computer controlled character, which is an intelligent opponent controlled by the Interaction Graph [2].

D. Buffer Size and Recognition Accuracy

We recorded a series of control signals from a player, and recognized the action with different sizes of the main buffer. For each buffer size, the results are classified into four groups: (1) recognized correctly (2) recognized correctly after an unnoticeable action switching (3) recognized correctly after a noticeable action switching (4) recognized incorrectly. (1) and (2) are acceptable results for our system, and although (3) is also acceptable for some games, here we employ a tougher quality requirement and consider them as unacceptable, together with (4).

First, we recorded a series of control signals performed by an experienced player who had played the game for more than 2 hours. The player randomly performed the actions for about one minute. Figure 8 shows the recognition rate with different sizes of the buffer. The X axis represents the number of frames of the main buffer and the Y axis represents the percentage of the recognition rate. The green solid line represents acceptable recognition, which is the sum of group (1) and (2). The red solid line represents unacceptable recognition, which is a sum of group (3) and (4). The dotted lines give extra information about the percentage of actions that resulted in switching. Since the player is experienced, the rate of acceptable recognition is high, and that of action switching is low. The accuracy drops significantly when the main buffer size is smaller than 6 frames. Such a value becomes a suitable trade-off between system responsiveness (1/10 second) and accuracy (>95%).

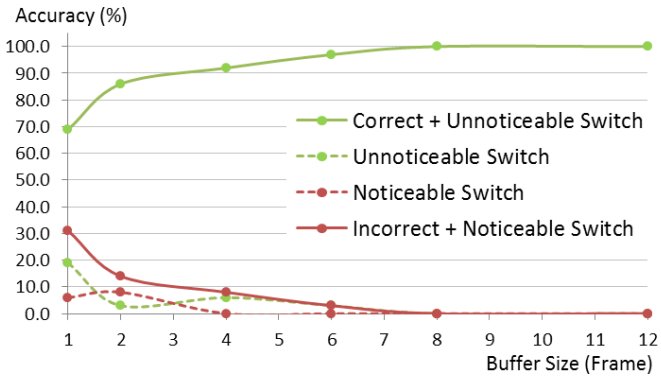


Fig. 8. The accuracy of action recognition with the control signals from an experienced player across different size of the main buffer.

We performed the same experiment with a novice player who has played the game for only 20 minutes. Figure 9 shows the results. We noticed that the actions performed by the novice player were more ambiguous, and the features of different classes of punches were not clearly performed. This leads to a lower recognition rate. However, even in this case, our system still performs better than traditional single buffer approaches thanks to the supplementary buffer and the intermediate switching. The supplementary buffer provides around 10% to 20% increase in the accuracy when the main buffer alone cannot recognize the actions correctly.

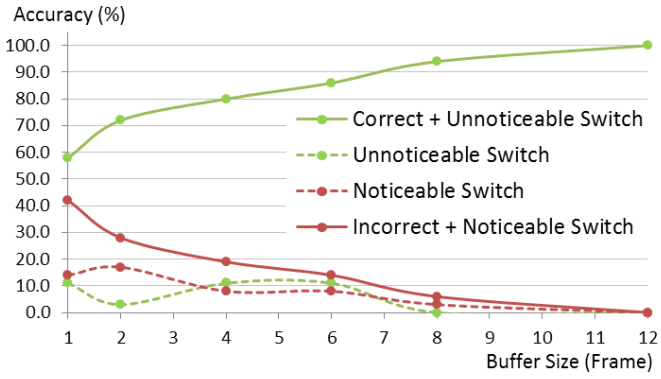


Fig. 9. The accuracy of action recognition with the control signals from a novice player across different size of the main buffer.

VII. Discussions

In this section, we provide further discussions on our system.

When matching the user signal with the signals in the database, instead of generating a mean signal for action matching, it is possible to match with all control signal samples, and find the most similar one. However, this is a lot more computationally costly, and does not significantly improve the matching accuracy, as the samples contain small variations only. Although the extra computational cost is not significant to high power CPU, we prefer using the mean signal such that our algorithm can be applied in game consoles.

DTW is the most computationally costly part in the action matching process. However, it is important to apply DTW instead of a simpler mapping such as uniform scaling because of the nature of the acceleration profile. For most actions, the acceleration profile consists of the accelerating part and the decelerating part. If the duration is not matched correctly, the positive acceleration part may be matched to the negative acceleration part (or vice versa), which will produce a large difference between the two profiles. As a result, the accuracy of the recognition decreases significantly without DTW.

We found that the signal profile can vary largely between left handed players and right handed players. Although the DTW eases the difficulty in matching the punching movements conducted by people of different handedness, we would suggest preparing two profiles for left and right hand players.

Our technique of motion matching shows significantly better performance comparing to those commonly used in accelerometer-based games. In terms of controllability, our system has a noticeably shorter response time than popular games such as the boxing game in the Wii Sports. Furthermore, we observed that in the Wii Sports boxing game, once an action is started, it will carry on until the end. It means that the system does not have a framework to switch the action even when the controllers are providing such information. Another observation from the Wii Sports boxing game is that it requires significantly longer time for synchronization, which can increase up to seconds. On the contrary, our system can mostly synchronize the user control and the character's action within 1/3 of a second.

The dual buffer framework presented in this paper is also applicable for sensors other than accelerometers, for example, optical trackers, magnetic trackers, and gyro sensors.

VIII. Conclusion and Future Works

We presented a dual buffer framework for fast action recognition. The main buffer stores the signals from the controllers for fast recognition. The supplementary buffer stores extra signals when they are available, and uses them to perform more accurate recognition. If the initial recognition is correct, we continue the current action. Otherwise, we switch to a newly recognized action. Using this framework, we can perform accurate character control with minimal control time lag. We demonstrate this framework using an accelerometer-based controller, and tested it with a boxing game in which the player can box with intelligent computer controlled characters.

As a future work, we would like to enhance the system by recognizing and modeling the details of the user performed actions. Currently, we can only classify some specific classes of actions, such as straight punches and upper cuts. Attributes such as the speed and the direction are not taken into account at the moment. We plan to make use of such data to edit the movements such that they appear more similar to the motion performed by the user.

Another future direction is to improve the intelligence of the computer controlled character to enhance the realism of the game. Currently, we apply the Interaction Graph, which is a machine learning approach, to control the virtual character. The control is only optimal when the opponent's actions are within expectation. If the user performs something unexpected, such as continuously avoidance, the computer controlled character will incorrectly predict the user's action and act sub-optimally. Enabling the computer player to adapt to different human players through a short observation of the player is an interesting research direction.

References

[1] H. P. H. Shum, T. Komura, and S. Yamazaki, "Simulating competitive interactions using singly captured motions," *Proceedings of ACM Virtual Reality Software Technology 2007*, pp. 65–72, 2007.

[2] H. P. H. Shum, T. Komura, and S. Yamazaki, "Simulating interactions of avatars in high dimensional state space," in *I3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games*. New York, NY, USA: ACM, 2008, pp. 131–138.

[3] J. Chai and J. K. Hodgins, "Performance animation from low-dimensional control signals," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 686–696, 2005.

[4] T. Shiratori and J. K. Hodgins, "Accelerometer-based user interfaces for the control of a physically simulated character," *ACM Trans. Graph.*, vol. 27, pp. 123:1–123:9, December 2008. [Online]. Available: http://doi.acm.org/10.1145/1409060.1409076

[5] A. Treuille, Y. Lee, and Z. Popovic', "Near-optimal character animation with continuous control," *ACM Transactions on Graphics*, vol. 26, no. 3, pp. 7:1–7:7, 2007.

[6] J. Lee and K. H. Lee, "Precomputing avatar behavior from human motion data," *Proceedings of 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 79–87, 2004.

[7] J. McCann and N. S. Pollard, "Responsive characters from motion fragments," *ACM Transactions on Graphics (SIGGRAPH 2007)*, vol. 26, no. 3, Aug. 2007.

[8] T. Graepel, R. Herbrich, and J. Gold, "Learning to fight," *Proceedings of Computer Games: Artificial Intelligence Design and Education (CGAIDE 2004)*, pp. 193–200, 2004.

[9] H. J. Shin and H. S. Oh, "Fat graphs: constructing an interactive character with continuous controls," in *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2006, pp. 291–298.

[10] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 473–482, 2002.

[11] R. Smith, "Open dynamics engine. www.ode.org," 2005.

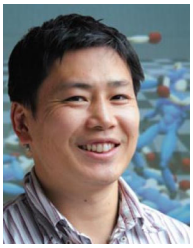
[12] T. Jakobsen, "Advanced character physics," *In Game Developers Conference Proceedings*, pp. 383–401, 2001.

[13] V. B. Zordan and J. K. Hodgins, "Motion capture-driven simulations that hit and react," *Proceedings of ACM SIGGRAPH Symposium on Computer Animation*, pp. 89 – 96, 2002.

[14] H. P. H. Shum, "Simulating interactions among multiple characters," Ph.D. dissertation, University of Edinburgh, UK, 2010.

[15] V. B. Zordan, A. Majkowska, B. Chiu, and M. Fast, "Dynamic response for motion capture animation," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 697–701, 2005.

Hubert P. H. Shum Hubert P. H. Shum is a lecturer in the University of Worcester. Before joining the university, he worked as a post-doctoral researcher in RIKEN Japan, as well as a research assistant in the City University of Hong Kong. He received his Ph. D. degree in the School of Informatics from the University of Edinburgh, and received his M. Sc. and B. Eng. degrees from the City University of Hong Kong. His research interests include character animation, machine learning and physical simulations.



Taku Komura Taku Komura is currently a Lecturer at School of Informatics, Edinburgh University, UK. He received his B. Sc., M. Sc. and D. Sc. in Information Science from the University of Tokyo. His research interests include topology-based motion synthesis and application of machine learning to character animation.

Shu Takagi Shu Takagi is a professor in the University of Tokyo and a team leader in CSRP, RIKEN Japan. He received his Doctor of Engineering from the University of Tokyo in 1995. Since then, he had worked as a research associate, a lecturer, as well as an associate professor in the same university. His research interests include fluid mechanics, computational biomechanics and medical ultrasound.