Body Motion Design and Analysis for Fighting Game Interface

Pujana Paliyawan[a,b], Kingkarn Sookhanaphibarn[a], Worawat Choensawat[a], and Ruck Thawonmas[b]

[a]Multimedia Intelligent Technology Lab, School of Science and Technology,
Bangkok University, Bangkok, Thailand
[b]Intelligent Computer Entertainment Lab, Graduate School of Information Science and Engineering,
Ritsumeikan University, Shiga, Japan

Email: pujana.p@gmail.com, kingkarn.s@bu.ac.th, worawat.c@bu.ac.th, ruck@is.ritsumei.ac.jp

***Abstract*—This paper presents a full-body motion-control game interface based on a Kinect device. A set of postures and motions for controlling game characters is presented, and the posture-detection algorithm for each posture is implemented by using a rule-based technique with rule-and-threshold optimization. Our experiments show that the proposed optimization can improve the accuracy of posture detection by 13.70%. The proposed techniques are applied to the fighting game called FightingICE, a game platform in recent CIG competitions.**

Keywords—Kinect; Motion and Posture Analysis; Detection and Classification; FightingICE; Variable Refinement;

I. Introduction

Today, there are many varieties of video games. A fighting game is a genre of games in which one or more players are engaged in a fighting combat. Traditionally, game characters are controlled via a gamepad and a keyboard; however, playing the video games with these devices has become a factor in reduced activity levels and childhood obesity risk [1].

In 2010, Kinect was invented by Microsoft as an input device for playing games by using gestures and spoken commands. It shook up the video-game experience and leaded to the physical interaction-gameplay market. Kinect has recently gained a lot of attention, not only from gamers, but also from researchers who work on posture recognition or physical activity promotion [2]. Unfortunately, the limitation is that the raw data captured by Kinect are joint position data, which by themselves cannot describe human postures. Therefore, there are needs in an approach for implementing a posture detection algorithm based on the Kinect data.

The goal of this paper is to develop a Kinect interface for fighting games. We propose a design of body postures and motions (a sequence of postures) for controlling a 2D fighting game character along with an approach for optimizing the global accuracy rate of the posture detection algorithm. Although the interface can be tested in any games with no requirement of accessing game source codes, FightingICE[1] [3] is selected as a pilot study in this experiment as it covers basic features in the genre of fighting games.

[1] www.ice.ci.ritsumei.ac.jp/~ftgaic/
[2] source: msdn.microsoft.com

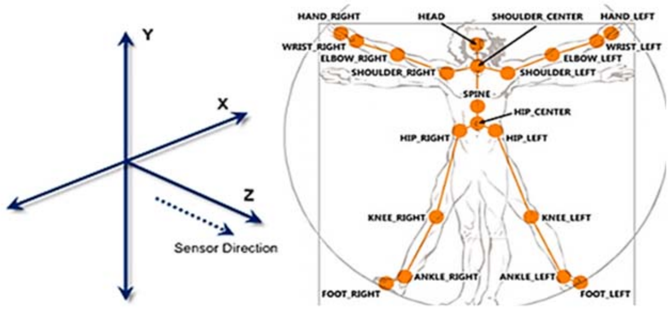


Fig. 1. Kinect skeleton data[2].

II. Background

A. Kinect

The raw input captured from Kinect is called "Skeleton Data [4]," which represent the positions of human 20 body joints each in a 3D coordinate (*X*, *Y*, *Z*) as shown in Fig. 1.

B. Kinect for Games

In the work of Pirovano et al. [5], Kinect was used for developing a rehabilitation game to recover balancing ability. The player was asked to play the game called "Fruit Catcher," in which he/she has to control a game character to move and catch falling fruits. The result showed that Kinect is an alternative way to support rehabilitation at home.

In Burelli et al.'s work [6], they investigated whether game players' experience can be predicted by their movements and in-game behaviors. Kinect was used to capture the head and torso positions of a player. The result showed that by the use of information about motions and in-game behaviors, it is possible to predict aspects of player experience such as perception of challenge.

Our work differs to the aforementioned work [5, 6] in that Pirovano et al. used Kinect to capture human body joints' positions for mapping them to a game avatar, but we aim at detecting full-body motions and encoding them into game input. For the work of Burelli et al., they focused on using Kinect for the purpose of user-experience analysis but not the game control.

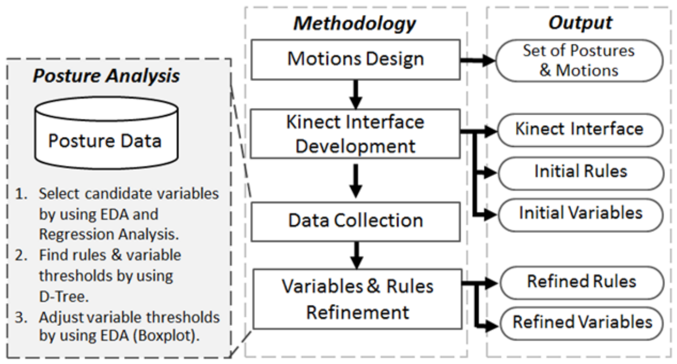

Posture Analysis
Posture Data
1. Select candidate variables by using EDA and Regression Analysis.
2. Find rules & variable thresholds by using D-Tree.
3. Adjust variable thresholds by using EDA (Boxplot).
Methodology
Motions Design
Kinect Interface Development
Data Collection
Variables & Rules Refinement
Output
Set of Postures & Motions
Kinect Interface
Initial Rules
Initial Variables
Refined Rules
Refined Variables

Fig. 2. Framework of our methodology.

III. Proposed Methodology

A. *Overview of our proposed methodology*

The proposed methodology (as shown in Fig. 2) begins with motion design and Kinect interface development, where the set of postures and motions are designed, and their detection algorithms are initialized by the developer's experiment and observation.

The next step is to optimize the performance of Kinect interface in the real environment. This step is done by collecting data from volunteers and analyzing them for discovering rules and variable thresholds that achieve the optimal global accuracy. The knowledge discovered is then used to refine the detection algorithms.

B. *Independent Variables*

Raw data acquired from Kinect themselves cannot describe human postures. Therefore, variables for posture classification are introduced in this work. Most of the variables individually represent the relative difference between two body joints. For example, *handL_sc_Zd* represents the difference in Z-Axis between Left Hand and Center Shoulder (*handL_Z–shoulderCenter_Z*). Examples of such variables are discussed in Section IV.

C. *Output Variables* (*Input Keys to the Game*)

Once the independent variables are analyzed, the interface will generate inputs to a targeted game. We propose five key inputs to cover all character movements as follows:

keyX: the horizontal movement command, where 0 is no command, 1 is the command to walk toward an opponent, and -1 is the command to step back toward the opponent. When *keyX* = 1, if the character is on the left side of the opponent, it will walk toward the right; else, it will walk toward the left.

keyY: the vertical movement command, where 0 is no command, 1 is to jump and -1 is to crouch.

keyXd: the quick move command, where 0 is no command, and 1 is to trigger double pressing of *keyX*. In other words, "*keyX* = 1 *& KeyXd* = 1" is to run toward the opponent, and "*keyX* = -1 *& KeyXd* = 1" is to dash backward.

keyAtk: the attack command, where 0 is no command, 1 is attack A, and 2 is attack B.

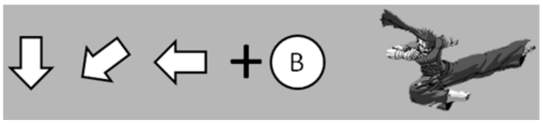


Fig. 3. Button combo for executing "Jump Kick" in FightingICE.

keySpecial: the command to execute a special skill (a move that originally requires a button combo for execution), where 0 is no command, and any value greater than 0 is a command for executing a special skill.

For example, "Jump Kick (the special skill #7)" is originally executed by a four-step button combo as shown in Fig. 3. Instead of requiring the player to perform the four-step posture combo when playing game with the proposed interface, the player only needs to perform one motion that is Knee Strike. Technically, this skill can be executed though the interface by setting *keySpecial* = 7; the interface will abandon detection of other keys and generate a sequence of inputs executing the four-step button combo automatically.

D. *Data Analysis*

Independent variables are analyzed and translated into output variables by the detection algorithms whose rules and variable thresholds are derived from data analysis using the following techniques;

1) *Exploratory Data Analysis* (*EDA*)

Exploratory data analysis [7] describes the basic features of data. The use of summaries and charts can help in examining a simple relationship between variables as well as in formulating hypotheses for further analysis. In this work, a boxplot is used to explore the data distribution and to find the optimal threshold of a variable of interest.

2) *Regression Analysis*

Regression analysis [7] is a technique in Statistics which is used to analyze the relationships among variables. In particular, it is used to examine whether variables of interest are affected by any other variables or not; in this work, we focus on variables that represent heterogeneous human characteristics.

3) *Decision Tree* (*D-Tree*)

In this work, decision trees [8] are used for motion classification based on the independent variables. A D-Tree output is a flowchart-like tree structure. It consists of a set of decision nodes and leaf nodes where a decision node indicates rules and thresholds of classification, and a leaf node shows a class outcome label.

IV. Motion Design

The design of all the postures and motions is based on the martial arts and combat practices [9-13]. In posture design, we also paid attention to issues of accuracy, meaningfulness and fun promotion. This section explains the meaning and the initial detection algorithm of each posture. Their technical details are available in Appendix.

A. *Base Posture*

When the interface starts up, the system will recognize the initial information of a player such as the heights of body joints (e.g. *ini_spine_Y* that is the height of spine in Y-Axis) and the

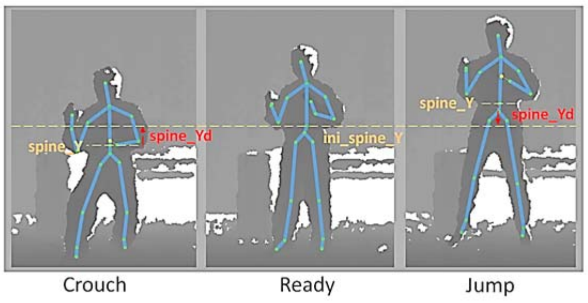


Fig. 4. Jump & Crouch postures.

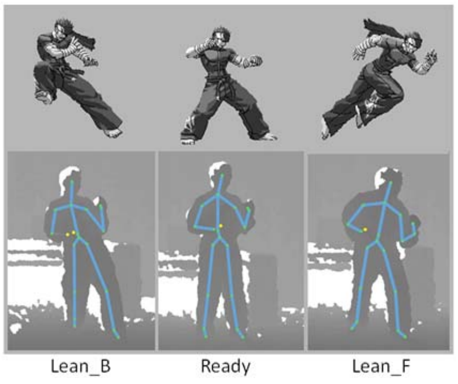


Fig. 5. Lean Forward & Lean Backward postures.

lengths of body parts (e.g. *ini_armlength* that is the Euclidian distance from the shoulder to the hand). These data are further used for detecting changes in the player's postures.

B. *Initial Postures Detection Algorithms*

1) *Jump & Crouch*

The postures Jump and Crouch are obviously matched with the movements of a game character.

In physical fitness assessment [14], it is considered that the standard height of jumping does not depend on the body size. Thus, Jump & Crouch detection is based on monitoring changes in the height of the player's spine in centimeters. The variable of interest here is the Y-coordinate of spine joint (*spine_Y*). The values of *spine_Y* for Jump and Crouch should be significantly different as shown in Fig. 4.

2) *Lean Forward & Lean Backward*

In martial arts, there is a common practice that the back of the fighter should be straight up, in order to support firmness in the standing posture [9-12]. The fighter should not lean, except when he/she is running or leaning back to evade or dodge the opponent's attack.

In the proposed system, leaning is used as a motion to execute "Quick Move (see Fig. 5)," which is that of running toward to the opponent when the player leans forward and dashing backward when the player lean backward. The leaning angle is the key variable in this detection, and it is calculated by using the mathematics in the spherical coordinate system.

3) *Step Forward & Step Back*

Stepping the right foot forward will trigger the character to walk toward the opponent, and stepping it backward will trigger the character to step back.

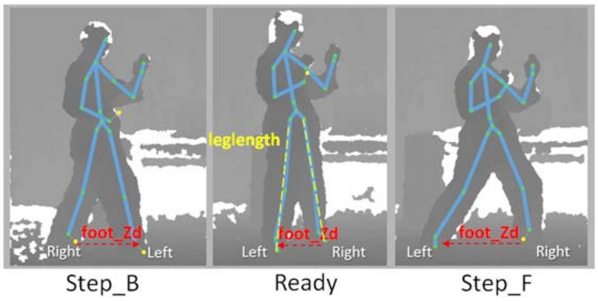


Fig. 6. Step Backward & Step Forward postures.

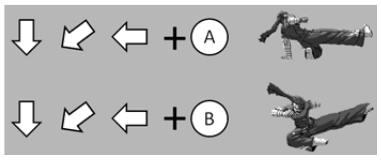


Fig. 7. Button combo for executing special skills #6-7 in FightingICE.

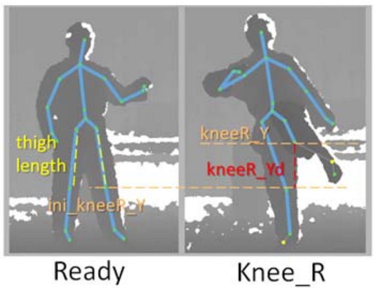


Fig. 8. Knee Strike posture.

The variable used for detection is called *Foot_Zd*, which represents the difference in the Z-axis between both feet (*Left Foot – Right Foot*). We assume that there is a tendency that the threshold value of each player is affected by the length of the legs (Fig. 6); hence *Foot_Zd* is represented as the percentage of the player's leg length (*Foot_Zd_per_leg*), instead of in centimeter.

4) *Left – Right Punch*

FightingICE consists of two basic attack buttons. The interface, therefore, uses the right punch motion for executing attack A (*keyAtk* = 1) and the left one for attack B (*keyAtk* = 2). The variable for detecting a right punch is called *handR_sc_Zd* (*Z of the right hand – Z of the shoulder*), and the threshold value for this variable is represented as the percentage of the player's arm length.

5) *Lean Forward with Right – Left Punch*

Punching while leaning forward is a posture used to execute special skills #4-5, which are required in a button combo in a traditional interface for execution. Its detection is based on the output of Lean Forward detection and Left – Right Punch detection.

6) *Left-Right Knee Strike*

Knee Strike is a posture used to execute special skills #6-7 (Fig. 7). Detection of this posture is based on the difference in the height of the knee between the base posture and a posture at a time of interest (Fig. 8). This posture can also be replaced by "Kick" posture as they share the same detection algorithm.

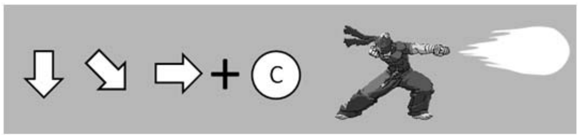


Fig. 9. Button for executing the ultimate attack in FightingICE.

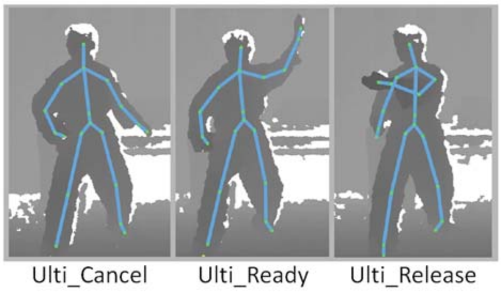


Fig. 10. Ultimate postures.

7) *Ultimate*

The motion for executing the ultimate skill (shooting a heavy projectile / the special skill #3 as shown in Fig. 9) is "Knifehand Strike," which is also known as Karate Chop or Hand Blade [13].

Execution of this motion consists of two steps (see Fig. 10). The first step is that of preparing the ultimate skill (*Ulti_Ready*) by raising the right-hand up over the head. The second step is that of chopping the hand down forward to shoot a heavy projectile to the opponent (*Ulti_Release*), after which the execution of the ultimate is completed. On the other hand, the player can cancel the current ultimate by lowering the hand down along the side of the body.

For the second step, it was found on the observation that while performing "Knifehand Strike," the right hand of the player will be the most fronted part of the body. Therefore, the posture is defined so as to release the ultimate when the right-hand moves down below the height of the head while it is in front of both knees. Otherwise the ultimate will be canceled.

Instead of Knifehand Strike, the ultimate skill can also be executed by some other motions that share the same detection algorithm. One possible alternative is "Spirit Bomb (Genki Dama in Dragon Ball[1])" which is the motion that raises the hands above the head to generate an energy ball and then throw it toward the opponent.

8) *Hadouken*

Hadouken[2] (also known as Surge Fist) is the most iconic skill from Street Fighter game series. This motion is executed by forming cupped hands to the side at the hip level, which generates energy in the palms and then shoots it toward the opponent. A similar motion that is also widely known is "Kamehameha[3]" from Dragon Ball.

In the proposed interface, shooting of a light projectile is executed by the Hadouken motion (the special skills #1-2). This motion consists of two steps: the be-ready step and the releasing/canceling step.

[1] thedaoofdragonball.com/blog/martial-arts/the-genki-dama-explained/
[2] streetfighter.wikia.com/wiki/Hadoken
[3] dragonball.wikia.com/wiki/Kamehameha

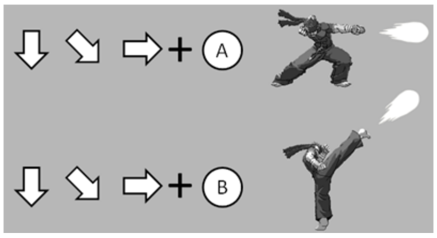


Fig. 11. Button for executing projectile attacks in FightingICE.

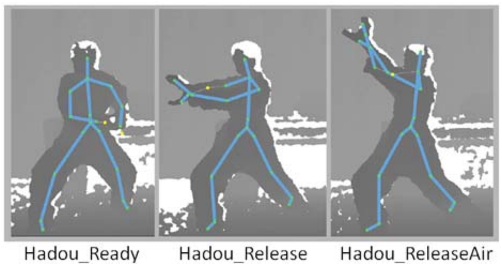


Fig. 12. Hahouken postures.

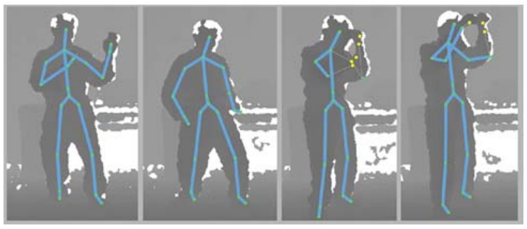


Fig. 13. Examples of Ready Stance postures.

The first step, *Ulti_Ready*, is to detect whether the left hand of the player is on the right side of the body (*X of the left hand – X of the spine* > 0) and below the height level of spine (*Y of the left hand – Y of the spine* < 0). The Euclidian between both hands needs to be within 30 cm.

Considering the second step, the system can detect if the player releases Hadouken (shoot a projectile) by simply detecting if *keyAtk* > 0 (Right Punch or Left Punch is detected). To classify whether Hadouken (Fig. 11-12) is shot straight forward (*Hadou_Release*) or shot into the air (*Hadou_ReleaseAir*), the relative difference between the left hand and the head over Y-Axis (*headL_head_Yd*) is considered. Hadouken is also cancelled when both hands are separated from each other greater than 30 cm.

9) *Ready Stance* (*Ready*)

Ready stances are those not in the aforementioned postures, and they send the zero-command to the game character.

There are various postures considered as Ready Stance (Fig. 13). For example, Horse Stance (Mabu in Chinese martial arts / Kiba-dachi in Karate) [9], L-stance (Niunja Sogi) in Taekwondo [10], Footwork in Boxing [11], and Standing Stance in Muay Thai [12].

V. Motion Analysis

In this section, posture data collected from volunteers are analyzed to discover thresholds and rules that optimize the global accuracy of the posture detection algorithms with respect to heterogeneous physical characteristics of players.

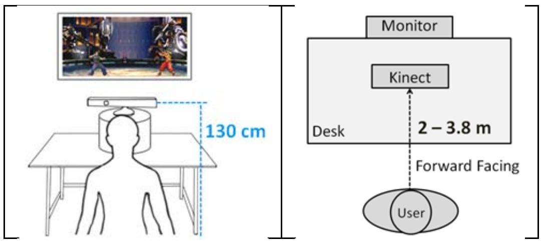


Fig. 14. Hardware setup for the experiment.

1. Stand → Jump → Stand → Jump
2. Stand → Crouch and remain still in the crouching posture for 3 seconds.
3. Change the standing position and repeat Step 1-2 again for 2 times.

Fig. 15. Data collection for Jump & Crouch.

1. Stand
2. Ultimate Ready (raise the right hand up) and remain still for 3 seconds.
3. Ultimate Release (chop the right hand down) and remain still for 3 seconds.
5. Change the standing point and repeat Step 1-2 again for 2 times.

Fig. 16. Data collection for Ultimate.

A. *Data Collection*

The proposed interface was tested with 18 players (13 males and 5 females). The average height of them was 166.94 ± 8.30 cm, and the average weight was 64.67 ± 16.55 kg. A Kinect device was placed at the height of 130 cm from the floor and about 2 to 3.8 meters away from the player as shown in Fig. 14.

Generally, the players were asked to perform a set of designed postures as stated in the previous section, three times per each posture, and remain still in a given posture for three seconds. They were then asked to change the standing position each time they perform the same motion to ensure accuracy regardless of the range between the player and the camera.

For "Jumping," each player was asked to jump six times in total as a replacement for remaining still for three seconds.

For "Ready Stance," as it consists of various gestures, each player was asked to perform the motion and move around the experiment area for 40 seconds. During this time, they can change their hand placement freely as in a real gameplay.

The procedure in collecting Jump & Crouch data is shown in Fig. 15, and the procedure in collecting the Ultimate data is depicted in Fig. 16.

B. *Data Preprocessing*

The data when each player was asked to remain still in each motion were extracted, and their class labels are marked up accordingly. Hence, the data while remaining still in a given posture are summarized to an instance of the data set.

The data set was then divided into sub-datasets, and the data analysis was performed on each group of class labels. In each analysis group, independent variables that seem not related to detection are removed. For example on punching detection, variables that tell nothing about user hands (e.g. *Foot_Zd*) are moved out. EDA and regression analysis is also used to formulate hypotheses on the relationship among variables.

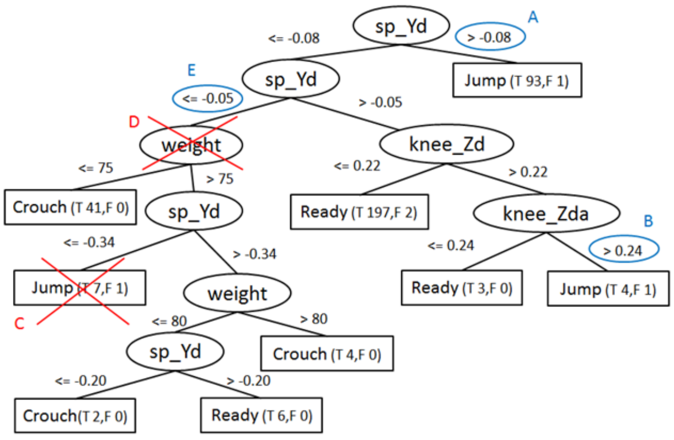


Fig. 17. D-Tree for Jump & Crouch detection.

Data with selected variables are then processed by D-Tree [2] to determine the rules on which variables should be used for detection of each posture. The thresholds of those variables are then adjusted by the exploratory analysis using boxplot.

C. *Refined Postures Detection Algorithms*

This section explains the detection algorithm of each posture after optimization.

1) *Jump & Crouch*

A group of class labels in this detection consists of *Ready*, *Jump* and *Crouch*. From the result of D-Tree (see Fig. 17), jumping is detected when the value of *spine_Yd* > 0.08 (Point A); if the height of the player's spine at any specific time is more than the spine height when he/she is standing, then the player is jumping. In case the value of *spine_Yd* ≤ 0.08 but still > -0.05, *knee_Zda* can be taken into consideration (Point B).

Point C in the figure can be interpreted as "When the height of the player's spine at any specific time is less than the spine height when he/she is standing, the player is jumping," This rule does not logically make any sense, so this point is trimmed.

Point D in the figure shows that crouching detection is affected by player's weight. However, since we found it contributes to only 6/204 of *Ready* detection and considered it unimportant, this variable can be trimmed. As the final result, crouching can be detected when *spine_Yd* ≤ -0.05.

2) *Lean Forward & Lean Backward*

A group of class labels in this detection consists of *Ready*, *Lean_F* and *Lean_B*. From the result, there is no change in the detection rules, and only variable thresholds are refined.

3) *Step Forward & Step Back*

A group of class labels in this detection consists of *Ready*, *Step_F* and *Step_B*. The data analysis starts by examining whether there is a relationship between the foot stepping range and the body size of the player. This process is done by performing the linear regression analysis by setting *Foot_Zd* as a dependent variable and setting *ini_leglength* and *height* as independent variables. The conducted regression analysis shows no relationship between the foot-stepping range and the player's body size at a 95% confidence interval; hence a variable *Foot_Zd_per_leg* is replaced by *Foot_Zd*.

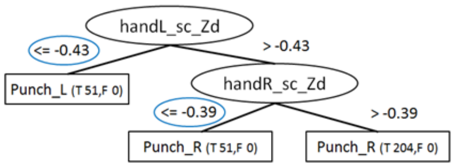


Fig. 18. D-Tree for Punching detection.

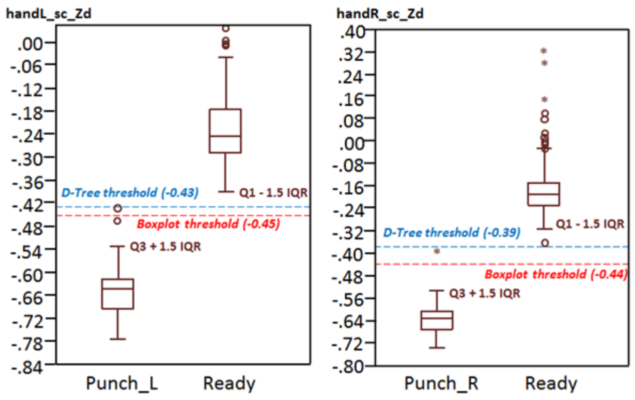


Fig. 19. Boxplot for Left-Right Punching detection.

4) *Left-Right Punch*

A group of class labels in this detection consists of *Ready*, *Punch_R* and *Punch_L*. The conducted regression analysis shows no relationship between the arm length and thresholds for punch detection; hence those variable thresholds are represented in a unit of meter instead of the percentage of arm length.

From the result of D-Tree (Fig. 18), the threshold values are -0.44 and -0.39. However, they are re-calculated by using Equation (1) where the parameters are obtained from the boxplot (Fig. 19), instead of using the values derived from D-Tree directly; this is to make them less sensitive to unseen data.

$$Threshold = \frac{(Q3+1.5\,IQR)_{ClassA}+(Q1-1.5IQR)_{ClassB}}{2} \quad (1)$$

5) *Left-Right Knee Strike*

A group of class labels in this detection consists of *Ready*, *Knee_R* and *Knee_L*. Again, the conducted regression analysis shows no relationship between the body size of the player and variable thresholds. Similar to punching detection; threshold values are recalculated based on boxplot.

6) *Ultimate Releasing & Cancelling Detection*

As the Ultimate consists of two steps for execution, the analysis here is divided into two subgroups.

The first subgroup is to detect be-ready step of the ultimate motion (raising the right hand). A group of class labels in this subgroup consists of *Ready* and *Ulti_Ready*.

The second subgroup is to detect whether the player decides to release or cancel the ultimate motion. A group of class labels here consists of *Ulti_Release* and *Ulti_Cancle*.

7) *Hadouken*

Similar to the ultimate motion, the analysis is divided into two subgroups.

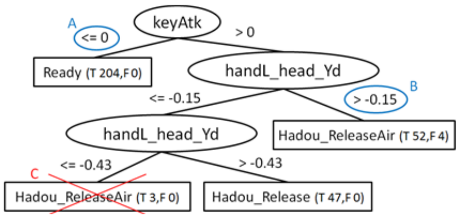


Fig. 20. D-Tree for Hadouken detection.

TABLE I. CHANGE IN THE RATE OF ACCURACY AFTER OPTIMIZATION

Motion	Initial	Refined	Change
Ready	96.57%	94.12%	-2.45%
Jump	89.22%	90.20%	0.98%
Crouch	74.51%	92.16%	17.65%
Lean_F	98.04%	100.00%	1.96%
Lean_B	45.10%	98.04%	52.94%
Step_F	58.82%	94.12%	35.29%
Step_B	47.06%	82.35%	35.29%
Punch_R	100.00%	100.00%	0.00%
Punch_L	100.00%	100.00%	0.00%
LeanPunch_R	60.78%	80.39%	19.61%
LeanPunch_L	58.82%	76.47%	17.65%
Knee_R	96.08%	100.00%	3.92%
Knee_L	100.00%	98.04%	-1.96%
Ulti_Ready	96.08%	97.06%	0.98%
Ulti_Release	100.00%	96.08%	-3.92%
Ult_Cancle	100.00%	100.00%	0.00%
Hadou_Ready	36.27%	93.14%	56.86%
Hadou_Release	84.31%	90.20%	5.88%
Hadou_ReleaseAir	64.71%	84.31%	19.61%
AVG	**79.28%**	**92.98%**	**13.70%**

The first subgroup is to detect Hadouken be-ready state. A group of class labels here consists of *Ready* and *Hadou_Ready*. According to D-Tree's results, if the left hand of the player is toward the right side of the body at least seven centimeters on the horizontal axis from the spine and is below at least 20 centimeters on the vertical axis from the center shoulder, then the player is doing Hadouken Readying.

The second subgroup is to detect Hadouken releasing state, in which releasing must also be classified by whether the player shoots the projectile in the forward direction or into the air. A group of class labels in the first subgroup consists of *Ready*, *Hadou_Release* and *Hadou_ReleaseAir*. From the D-Tree result as shown in Fig. 20, the system can detect whether the player releases Hadouken or not by simply detecting an attack key (Point A).The variable *headL_head_Yd* (relative difference in the Y axis between the left hand and the head) is for classifying the releasing direction (Point B).

In addition, Hadouken is canceled when both hands are separated from each other greater than 30 cm.

VI. RESULTS & DISCUSSION

After rules and thresholds of the detection algorithms are refined, the overall rate of accuracy is increased 79.28% to 92.98% (see TABLE I). Motions most benefited from optimization are "Leaning Backward" and "Hadouken Readying," which have low accuracy rates with the initial detection algorithms. The accuracies in detecting some motions are slightly dropped due to false positive on other motions. They can be further improved by introducing a technique to sort detection priority (which action should be taken in case more than one posture is detected at the same time).

In addition, we found that the garment type of player clothing can severely affect Kinect skeleton tracking capability. The worst case was found in Subject#18 where the Kinect completely failed to detect the player's elbows and hands. As a result, we decided to remove this player's data from the experiment and results reported in the paper.

VII. CONCLUSIONS AND FUTURE WORK

We proposed a motion and posture design for controlling a 2D fighting game and an approach for optimizing detection algorithms. The accuracy was significantly improved after the refinement, so we recommended that rule-and-threshold optimization by using training data should be taken as an essential process in user interface development.

An example video clip[1] shows that the proposed interface can control a fighting game character effectively. The purposed interface can possibly be applied to other existing games and not only limited to the genre of fighting games, which we plan to cover in the future.

REFERENCES

[1] E.A. Vandewater, M.S. Shim and A.G. Caplovitz, "Linking obesity and activity level with children's television and video game use," J Adolesc. 2004, vol. 27, no. 1, pp. 71-85

[2] H. Leutwyler et al., "Videogames to promote physical activity in older adults with schizophrenia," GAMES FOR HEALTH: Research, Development, and Clinical Applications 1.5, 2012, pp. 381-383.

[3] F. Lu, K. Yamamoto, L.H. Nomura, S. Mizuno, Y-M. Lee, and R. Thawonmas, "Fighting Game Artificial Intelligence Competition Platform," Proc. of the 2nd IEEE Global Conference on Consumer Electronics (GCCE 2013), Tokyo, Japan, Oct. 1-4, 2013, pp. 320-323.

[4] J. Webb, and J. Ashley, Beginning Kinect programming with the Microsoft Kinect SDK, [E-book], Apress©, pp. 1-321.

[5] M. Pirovano, R. Mainetti, G. Baud-Bovy, PL Lanzi, N.A. Borghese, "Self-adaptive games for rehabilitation at home," Proc. of 2012 IEEE Computational Intelligence and Games (CIG 2012), Granada, Spain, Sep. 11-14, 2012, pp. 179-186.

[6] P. Burelli, G Triantafyllidis, I Patras, "Non-invasive Player Experience Estimation from Body Motion and Game Context," Proc. of 2014 IEEE Conference on Computational Intelligence and Games (CIG 2014), Dortmund, Germany, Aug. 26-29, 2014, pp. 92-98.

[7] D.A. Freedman, "Statistical Models: Theory and Practice," Cambridge University Press, 2nd Edition, April 27, 2009, pp. 1-458.

[8] J. Han, M. Kamber, and J. Pei, "Data Mining Concepts and Techniques," Morgan Kaufmann Publishing, 3rd Edition, 2012, pp.1-744.

[9] R. Hill, "World of Martial Arts : The History of Martial Arts," [E-book], Lulu Publishing, Sep 8, 2008, pp. 1-139.

[10] C.H. Hi, "Taekwon-Do: The Condensed Encyclopedia," International Taekwon-Do Federation, 4th edition, 1995, pp. 1-765.

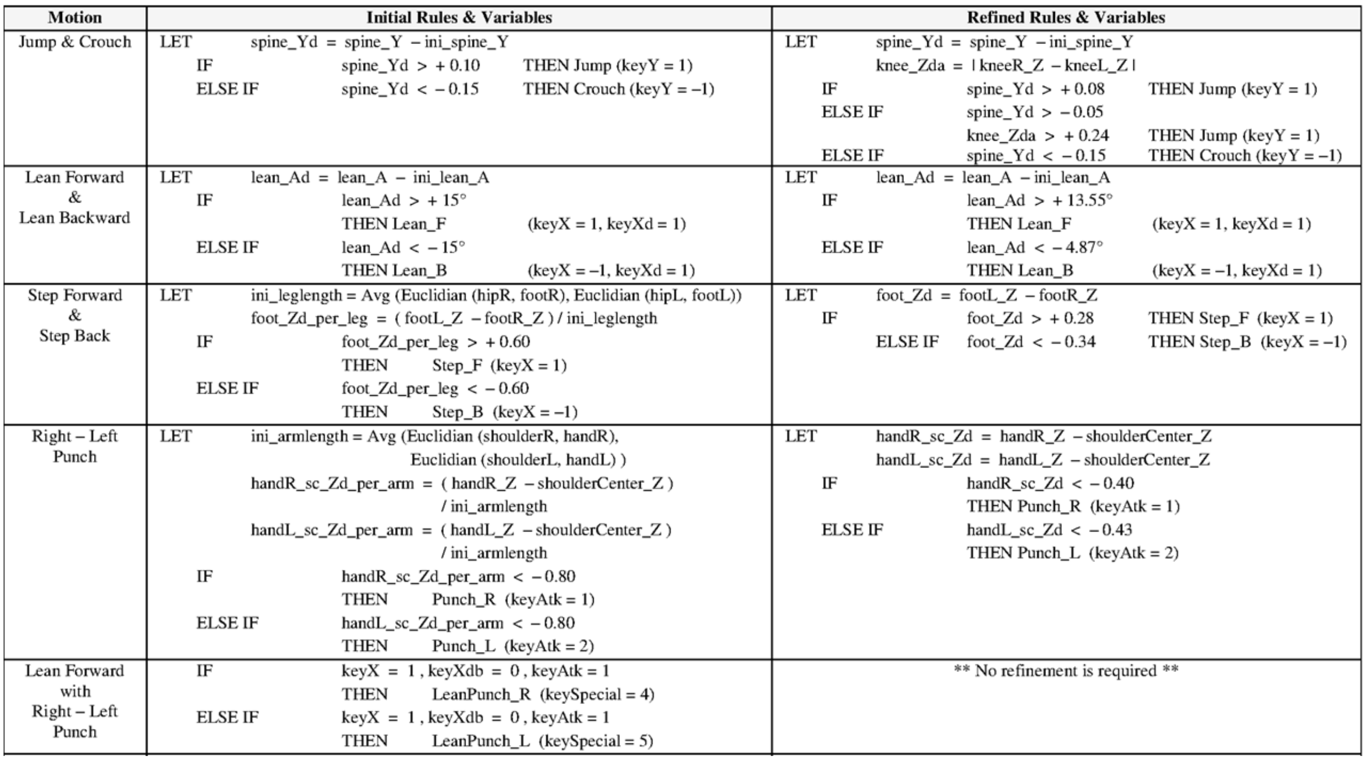
[11] C. Wigle, "Boxing Science Skills Book: A Reference Guide for Boxing Beginners," [E-book], Smashwords Edition, 2011, pp.1-36.

[12] Y. Ruerngsa, K.K. Charuad and J. Cartmell, "Muay Thai: The Art of Fighting," [E-book], 2008, pp. 1-277.

[13] E. Arus, "Biomechanics of Human Motion: Applications in the Martial Arts," CRC Press, December 13, 2012, pp. 1-559.

[14] The Cooper Institute, "Protocol for Anaerobic Power Testing," [Online], Available : www.tucsonaz.gov/files/police/CooperStandards.pdf

APPENDIX (DETECTION RULES & VARIABLES)



Motion	Initial Rules & Variables	Refined Rules & Variables

Jump & Crouch	LET spine_Yd = spine_Y − ini_spine_Y
	IF spine_Yd > + 0.10 THEN Jump (keyY = 1)
	ELSE IF spine_Yd < − 0.15 THEN Crouch (keyY = −1)
		LET spine_Yd = spine_Y − ini_spine_Y
		knee_Zda = ∣ kneeR_Z − kneeL_Z ∣
		IF spine_Yd > + 0.08 THEN Jump (keyY = 1)
		ELSE IF spine_Yd > − 0.05
		knee_Zda > + 0.24 THEN Jump (keyY = 1)
		ELSE IF spine_Yd < − 0.15 THEN Crouch (keyY = −1)

Lean Forward
&
Lean Backward
	LET lean_Ad = lean_A − ini_lean_A
	IF lean_Ad > + 15°
	THEN Lean_F (keyX = 1, keyXd = 1)
	ELSE IF lean_Ad < − 15°
	THEN Lean_B (keyX = −1, keyXd = 1)
		LET lean_Ad = lean_A − ini_lean_A
		IF lean_Ad > + 13.55°
		THEN Lean_F (keyX = 1, keyXd = 1)
		ELSE IF lean_Ad < − 4.87°
		THEN Lean_B (keyX = −1, keyXd = 1)

Step Forward
&
Step Back
	LET ini_leglength = Avg (Euclidian (hipR, footR), Euclidian (hipL, footL))
	foot_Zd_per_leg = (footL_Z − footR_Z) / ini_leglength
	IF foot_Zd_per_leg > + 0.60
	THEN Step_F (keyX = 1)
	ELSE IF foot_Zd_per_leg < − 0.60
	THEN Step_B (keyX = −1)
		LET foot_Zd = footL_Z − footR_Z
		IF foot_Zd > + 0.28 THEN Step_F (keyX = 1)
		ELSE IF foot_Zd < − 0.34 THEN Step_B (keyX = −1)

Right – Left
Punch
	LET ini_armlength = Avg (Euclidian (shoulderR, handR),
	Euclidian (shoulderL, handL))
	handR_sc_Zd_per_arm = (handR_Z − shoulderCenter_Z)
	/ ini_armlength
	handL_sc_Zd_per_arm = (handL_Z − shoulderCenter_Z)
	/ ini_armlength
	IF handR_sc_Zd_per_arm < − 0.80
	THEN Punch_R (keyAtk = 1)
	ELSE IF handL_sc_Zd_per_arm < − 0.80
	THEN Punch_L (keyAtk = 2)
		LET handR_sc_Zd = handR_Z − shoulderCenter_Z
		handL_sc_Zd = handL_Z − shoulderCenter_Z
		IF handR_sc_Zd < − 0.40
		THEN Punch_R (keyAtk = 1)
		ELSE IF handL_sc_Zd < − 0.43
		THEN Punch_L (keyAtk = 2)

Lean Forward
with
Right – Left
Punch
	IF keyX = 1 , keyXdb = 0 , keyAtk = 1
	THEN LeanPunch_R (keySpecial = 4)
	ELSE IF keyX = 1 , keyXdb = 0 , keyAtk = 1
	THEN LeanPunch_L (keySpecial = 5)
		** No refinement is required **

[1] bit.ly/CIG2015-ICELab

Right – Left Knee Strike	LET ini_thighlength = Avg (Euclidian (hipR, kneeR),
	Euclidian (hipL, kneeL))
	kneeR_Yd_per_thigh = (kneeR_Y − ini_kneeR_Y)
	/ ini_thighlength
	kneeL_Yd_per_thigh = (kneeL_Y − ini_kneeL_Y)
	/ ini_thighlength
	IF kneeR_Yd_per_thigh > + 0.33
	THEN Knee_R (keySpecial = 6)
	ELSE IF kneeL_Yd_per_thigh < + 0. 33
	THEN Knee_L (keySpecial = 7)
		LET kneeR_Yd = kneeR_Y − ini_kneeR_Y
		kneeL_Yd = kneeL_Y − ini_kneeL_Y
		IF kneeR_Yd > + 0.11
		THEN Knee_R (keySpecial = 6)
		ELSE IF kneeR_Yd > + 0.11
		THEN Knee_R (keySpecial = 7)

Ultimate	LET handR_head_Yd = handR_Y − head_Y
	handR_kneeR_Zd = handR_Z − shoulderCenter_Z
	handR_kneeL_Zd = handL_Z − shoulderCenter_Z
	Step1
	IF handR_head_Yd > + 0.15
	THEN Ulti_Ready (Go to Step2)
	Step2 :
	WHEN handR_head_Yd < 0
	IF handR_kneeR_Zd < 0
	handR_kneeL_Zd < 0
	THEN Ulti_Release (keySpecial = 3)
	ELSE Ulti_Cancel (Go back to Step 1)
		LET handR_head_Yd = handR_Y − head_Y
		handR_sc_Zd = handR_Z − shoulderCenter_Z
		Step1
		IF handR_head_Yd > + 0.10
		THEN Ulti_Ready (Go to Step2)
		Step2 :
		WHEN handR_head_Yd < 0
		IF handR_sc_Zd < − 0.35
		THEN Ulti_Release (keySpecial = 3)
		ELSE Ulti_Cancel (Go back to Step 1)

Hadouken	LET handL_sp_Xd = handL_X − spine_X
	handL_ sp _Yd = handL_Y − spine _Y
	headL_sc_Yd = handL_Y − shoulderCenter_Y
	handLR_ eu = Euclidian (handL_Y, headR_Y)
	Step1
	IF handL_sp_Xd > 0.00
	handL_sp_Yd < 0.00
	handLR_ eu < − 0.20
	THEN Hadou_Ready (Go to Step2)
	Step2 :
	WHEN keyAtk > 0
	IF headL_sc_Yd > 0
	THEN Hadou_ReleaseAir (keySpecial = 2)
	ELSE Hadou_Release (keySpecial = 1)
	
	WHEN handLR_eu > + 0.30
	THEN Hadou_Cancel (Go back to Step1)
		LET handL_sp_Xd = handL_X − spine_X
		handL_sc_Yd = handL_Y − shoulderCenter_Y
		headL_head_Yd = handL_Y − head_Y
		handLR_ eu = Euclidian (handL_Y, headR_Y)
		Step1
		IF handL_sp_Xd > + 0.07
		handL_sc_Yd < − 0.20
		keyAtk = 0
		THEN Hadou_Ready (Go to Step2)
		Step2 :
		WHEN keyAtk > 0
		IF headL_head_Yd > + 0.15
		THEN Hadou_ReleaseAir (keySpecial = 2)
		ELSE Hadou_Release (keySpecial = 1)
		
		WHEN handLR_eu > + 0.30
		THEN Hadou_Cancel (Go back to Step1)