SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 201X; 00:1– 5
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe

UKI: Universal Kinect-type-controller by ICE Lab

Pujana Paliyawan*,† and Ruck Thawonmas

Intelligent Computer Entertainment Lab,
Graduate School of Information Science and Engineering,
Ritsumeikan University, Japan

SUMMARY

Universal Kinect-type-controller by ICE Lab (UKI, pronounced as “You-key”) was developed to allow users to control any existing application by using body motions as inputs. The middleware works by converting detected motions into keyboard and/or mouse-click events, and sending them to a target application. This paper presents the structure and design of core modules, along with examples from real cases to illustrate how the middleware can be configured to fit a variety of applications. We present our designs for interfaces that decode all configuration details into a human-interpretable language, and these interfaces significantly promote user experience and eliminate the need for programming skill. The performance of the middleware is evaluated on fighting-game motion data, and we make the data publicly available so that they can be used in other researches. UKI welcomes its use by everyone without any restrictions on use; for instance, it can be used to promote healthy life through a means of gaming and/or used to conduct serious research on motion systems. The middleware serves as a shortcut in the development of motion applications – coding of an application to detect motions can be replaced with simple clicks on UKI.

KEY WORDS: Middleware; Motion controller; Kinect; User interfaces; Video games.

1. INTRODUCTION

Kinect, one of the most famous motion controllers, is well-known as a powerful device for promoting healthy life through a means of game playing. Besides gaming, this device has gone viral in various fields of applications. However, publishing a Kinect application is still not appealing to developers as it needs more time and effort, yet is not more profitable than keyboard applications [1]. Year-by-year ever fewer Kinect games are released into the market; they are usually of poor quality and a narrow variety of genres makes motion gaming reach just a niche group of gamers. As a result, the demand for motion controllers has declined and the death of motion gaming has been signaled [2].

As a solution, we introduce a new motion application development philosophy: “Rather than developing specific applications for motion controllers, motion control middleware should be developed to welcome the use of motion controllers with existing applications.” Accordingly, we propose a middleware framework that allows users to play any games or control any applications by using body motion as input. The middleware in this paper is an implementation of concepts discussed in our previous studies [3, 4]. The main contributions of this work are:

*Correspondence to: P. Paliyawan, Intelligent Computer Entertainment Lab, Graduate School of Information Science and Engineering, Ritsumeikan University, Japan.

†E-mail: Pujana.P@gmail.com

- Designs for modules and processes that make the middleware flexible enough to cover control specifications in most existing games.
- Designs for GUIs in a human-interpretable language that allow anyone to configure the middleware for playing games or conducting Kinect research projects without the need for programming skill.
- Evaluation of the detection accuracy on motion models used by the middleware.

2. RELATED WORK

According to our survey, few groups have worked on building middleware for motion control. Middleware that is publicly known and available for download includes the following:

- ***FAAST*** [5] introduced by Suma et al., is middleware that facilitates integration of motion control with virtual reality and game applications. Similarly to ours, their middleware works by converting detected motions into keyboard and/or mouse-clicks events, and sending them to a target application.
- ***Kinect2Scratch*** [6] is middleware that allows data from the Kinect be sent to Scratch applications. It works by adding a new type of input to the Scratch Project Editor. The user can modify the game source code for playing with Kinect by simply mapping the movement of a body joint to that of an object in the game.

The above are successful middleware products. However, some essential features are still not available. Therefore, they cannot handle the control specifications in some genres of games, such as fighting games, effectively (discussed in the next section).

3. SOFTWARE OUTLINE

The Universal Kinect-type-controller by ICE Lab (UKI, pronounced as “You-key”) was conceptually introduced in our previous studies [3, 4] as middleware for playing games (Figure 1). Even though it can be used with any application, we focus in this paper on gaming usage as it is the most sophisticated usage of Kinect.

3.1. Middleware overview

During gameplay (Figure 2), the middleware reads streaming skeleton data, detects motions, converts them into keyboard and/or mouse-clicks events and sends to a target game application based on procedures written in a predefined MAP file.

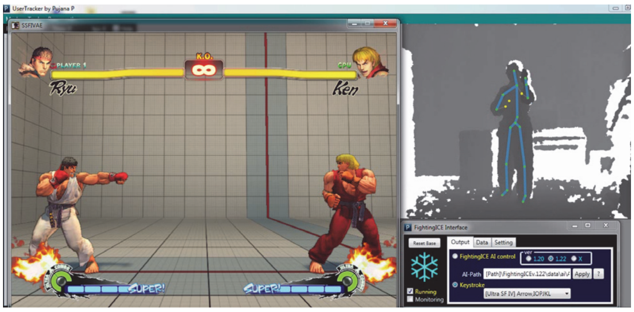


Figure 1. Use of UKI on Ultra Street Fighter IV [7].

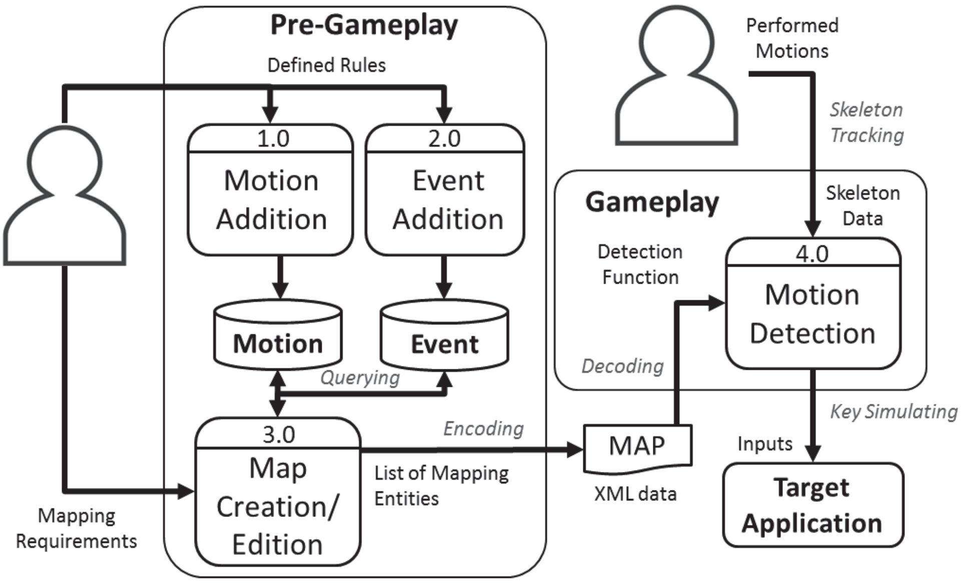


Figure 2. Overview of the middleware.

The MAP file, an abbreviation for "Conditions-to-Events Mapping file", contains mapping components. The most primitive mapping component is *Detection* that contains a list of conditions (*List<IF>*), used mainly for describing how to detect a motion, and a list of events (*List<THEN>*), used mainly for describing what to do when a motion is detected. An example of *Detection* "Right Punch" is the following.

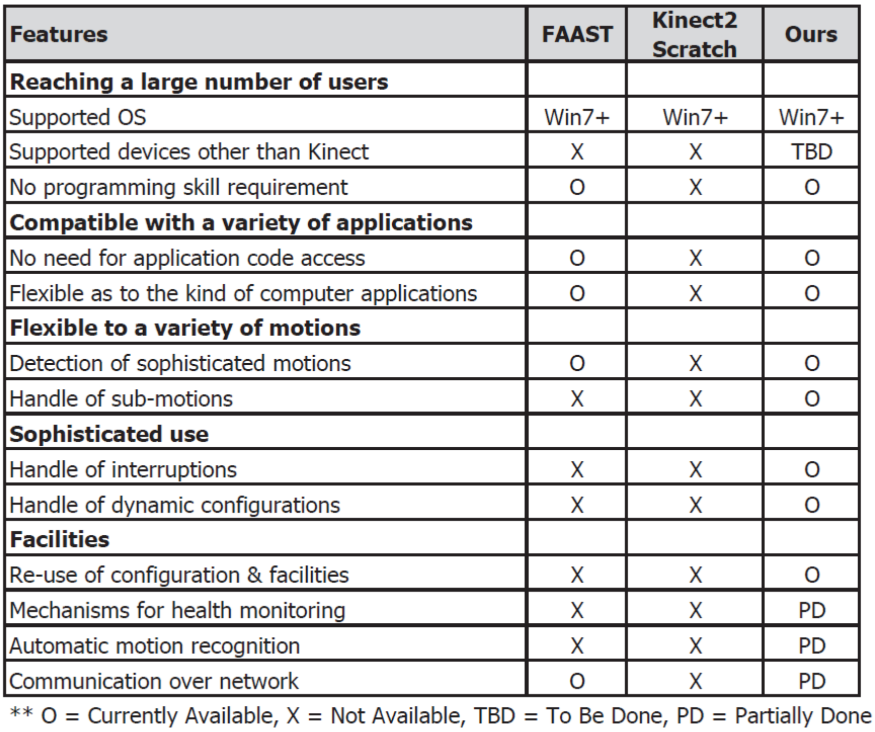
- *List<IF>*: Right Hand is in front of Center Shoulder more than 35 cm.
- *List<THEN>*: Send a command to hold key Z until the end-of-motion (EoM), in other words, until the time when "Right Punch" is no longer detected.

The MAP file is prepared before gameplay. Both lists can be either directly written into the MAP file in this process or independently added into their databases (Process 1.0 and 2.0) and consequently queried by using a special type of conditions (discussed in 4.1.3) or that of events (discussed in 4.1.4).

3.2. Middleware design

Our subgoals (Table 1) and how we achieve them are explained in this subsection.

Table I. Designed features of the middleware.



Features	FAAST	Kinect2 Scratch	Ours
Reaching a large number of users			
Supported OS	Win7+	Win7+	Win7+
Supported devices other than Kinect	X	X	TBD
No programming skill requirement	O	X	O
Compatible with a variety of applications			
No need for application code access	O	X	O
Flexible as to the kind of computer applications	O	X	O
Flexible to a variety of motions			
Detection of sophisticated motions	O	X	O
Handle of sub-motions	X	X	O
Sophisticated use			
Handle of interruptions	X	X	O
Handle of dynamic configurations	X	X	O
Facilities			
Re-use of configuration & facilities	X	X	O
Mechanisms for health monitoring	X	X	PD
Automatic motion recognition	X	X	PD
Communication over network	O	X	PD

** O = Currently Available, X = Not Available, TBD = To Be Done, PD = Partially Done

3.2.1. Middleware design

- ***Supported OS and devices***: Because Microsoft Kinect SDK v1.8 [8] was selected; UKI supports Windows 7 or above and thus covers more than 80% of Windows users [9]. UKI is expected to support a variety of devices in the future.

- ***No programming skill requirement***: On each of the GUIs, configuration details are represented in a human-interpretable language, allowing anyone to design conditions for detecting motions in a human friendly fashion.

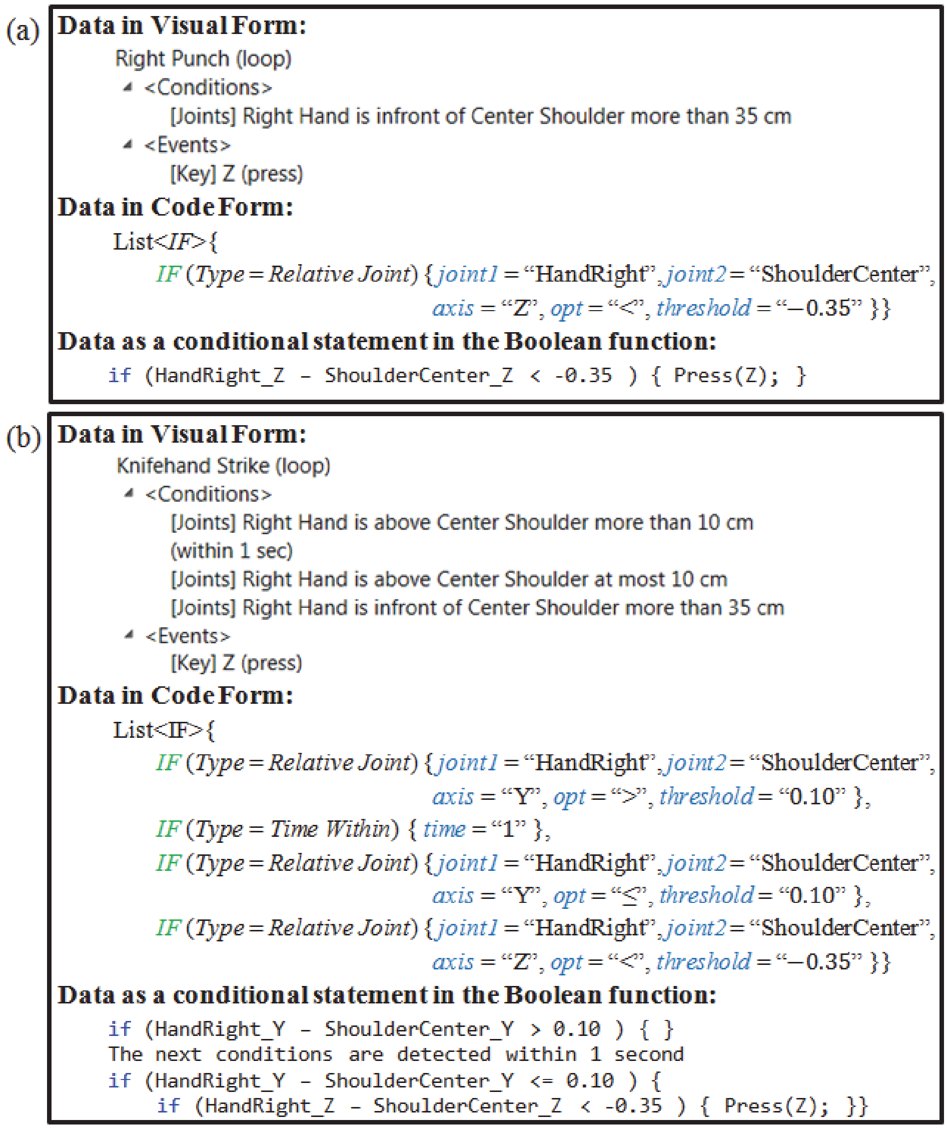
3.2.2. Compatible with a variety of applications

- ***No need for application code access***: The middleware can work with any applications without accessing their source codes.

- ***Flexible as to the kind of computer applications***: The middleware can work with applications written in any programming languages and using any APIs, and compatible with applications installed by any means.

3.2.3. Flexible to a variety of motions

- ***Detection of sophisticated motions***: To check whether a motion is detected, skeleton data are fed into a Boolean function formed by a list of conditions (Figure 3). "Right Punch" is considered as a basic motion, a single-step motion with only one condition. In contrast to basic motions, sophisticated motions are those consisting of multiple conditions and usually of multiple steps (e.g., Knifehand Strike [3]). In order to detect sophisticated motions, there are two major requirements: (1) the data representation is capable of representing a Boolean function and (2) the stage of motion is traceable.



(a)

Data in Visual Form:

```
Right Punch (loop)
  ◢ <Conditions>
      [Joints] Right Hand is infront of Center Shoulder more than 35 cm
  ◢ <Events>
      [Key] Z (press)
```

Data in Code Form:

```
List<IF>{
    IF (Type = Relative Joint) {joint1 = "HandRight", joint2 = "ShoulderCenter",
                               axis = "Z", opt = "<", threshold = "−0.35" }}
```

Data as a conditional statement in the Boolean function:

```
if (HandRight_Z - ShoulderCenter_Z < -0.35 ) { Press(Z); }
```

(b)

Data in Visual Form:

```
Knifehand Strike (loop)
  ◢ <Conditions>
      [Joints] Right Hand is above Center Shoulder more than 10 cm
      (within 1 sec)
      [Joints] Right Hand is above Center Shoulder at most 10 cm
      [Joints] Right Hand is infront of Center Shoulder more than 35 cm
  ◢ <Events>
      [Key] Z (press)
```

Data in Code Form:

```
List<IF>{
    IF (Type = Relative Joint) {joint1 = "HandRight", joint2 = "ShoulderCenter",
                               axis = "Y", opt = ">", threshold = "0.10" },
    IF (Type = Time Within) { time = "1" },
    IF (Type = Relative Joint) {joint1 = "HandRight", joint2 = "ShoulderCenter",
                               axis = "Y", opt = "≤", threshold = "0.10" },
    IF (Type = Relative Joint) {joint1 = "HandRight", joint2 = "ShoulderCenter",
                               axis = "Z", opt = "<", threshold = "−0.35" }}
```

Data as a conditional statement in the Boolean function:

```
if (HandRight_Y - ShoulderCenter_Y > 0.10 ) { }
The next conditions are detected within 1 second
if (HandRight_Y - ShoulderCenter_Y <= 0.10 ) {
    if (HandRight_Z - ShoulderCenter_Z < -0.35 ) { Press(Z); }}
```

Figure 3. A condition for detecting "Right Punch (a)" and "Knifehand Strike (b)" in various forms (more details on them are discussed in 4.1).

In Kinect2Scratch, the data representation in use is "3D Joint Positions [10]," that is based on raw Kinect data. This form of data is not suitable for constructing a Boolean function because values vary according to the user's standing position, and merely one joint of the body alone cannot describe human motions.

In both FAAST and UKI, the main data representation used is "Relative Positions of Joints [10]," which represents the relative difference between two body joints and is highly suitable to form a Boolean function. In addition, mechanisms for monitoring the stage of motion are added (discussed in 4.2).

- ***Handle of sub-motions***: When motion A contains motion B as its component, it is said that motion B is a sub-motion of A, or equivalently A is a super-motion of B. For example, let us assume that there are three *Detection*s in a MAP file:

the first is the one in 3.1 denoted as [Right Punch → Z], while the second and the third are [Left Punch → X] and [Two-Handed Punch → C], respectively. "Two-Hand Punch" is a superset motion of "Right Punch" and "Left Punch." If the player performs "Two-Hand Punch" with the intention to send key "C," not only "C" but "Z" and "X" will also be sent.

In FAAST, the rule of processing is that each *Detection* is independent: as long as all conditions in its *List<IF>* are satisfied, all events in its *List<THEN>* will be processed. This corresponds to the expectation that multiple motions can be performed simultaneously to create a combination of outputs, such as~~like~~ punching (sends "Z") while jumping (sends "▲") to create the combination of keys "▲+ Z" for executing "Jump Attack" in the game. Attention must be given to avoid combining outputs from a super-motion and its sub-motion(s).

In UKI, another type of component called *DetectionGroup* is at the top level of the hierarchy of the MAP file (discussed in 4.1). Outputs from different *DetectionGroups* can be combined, whereas only one *Detection* among multiple *Detections* can be fired at a time within a *DetectionGroup*. Therefore, only the desired outputs will be sent while performing a super-motion by placing a super-motion before its sub-motion(s) in their *DetectionGroup*.

3.2.4. Sophisticated use

The following are special requirements in some applications.

- ***Handle of interruptions***: In FAAST and UKI, events in each *Detection* are processed by running a thread. The thread is run for sending a sequence of key presses over time. One such a thread is that of sending a button combo to execute Hadouken [11] (Figure 4(a)). If, while the Hadouken thread is sending the button combo, another thread sends its outputs simultaneously, the sequence of key presses will be interrupted, which causes the execution of Hadouken to be failed (Figure 4(b))—this situation is called "interruption."

  In UKI, the aforementioned *DetectionGroup* can be used to prevent an interruption from other *Detections* within the same group, but not from different groups. For example, the *DetectionGroup* can guarantee that only one attack skill will be executed at a time, but it does not guarantee that there will be no other actions besides attack skills, such as~~like~~ jumping. In order to ensure that the expected outputs be sent one at a time, we, therefore, introduce an optional property of the *Detection* component called "Priority Process" (discussed in 4.3).

- ***Handle of dynamic configurations***: Before playing the game, the MAP file built for that game is fetched, and its mapping components are converted into configurations that instruct the middleware how to detect motions and what to do when each of them is detected. However, there are cases where changing some configurations is demanded on-the-fly during gameplay—such as in *Street Fighter* where the expected keys for executing button combos are subject to change according to the player character's facing direction (Figure 5). Such configurations capable of changing their contents during gameplay are called "dynamic configurations."

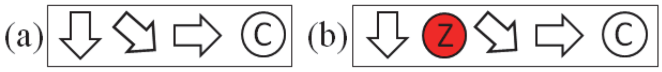


Figure 4. (a) The expected button combo for execution of Hadouken, (b) A failed execution of Hadouken—dues to an interruption.

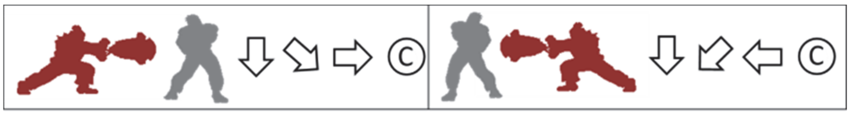


Figure 5. Difference in a button combo for execution of Hadouken on two different facing directions of the player character (red).

In FAAST, a MAP file is fetched once before gameplay. All configurations remain static during the gameplay. Therefore, dynamic configurations cannot be handled.

In UKI, the need for dynamic configurations is handled by two proposed MAP components: (1) *Key Replacement* event and (2) *Variables*. *Key Replacement* event (4.1.4) is an efficient way to shift commands from one key to another key, such as reversing between the left arrow button and the right one to handle the facing direction in 2D games. *Variables* (4.4) are used as switches for turning on/off some configurations of the MAP file.

3.2.5. Facilities

- ***Re-use of configurations & facilities***: Preparation of the MAP file can be time-consuming because conditions for detecting every single motion must be given. It is preferable if the user can reuse and/or modify some motions that have already been defined. Therefore, we introduce the Motion Database and Event Database for keeping sets of conditions and events. These databases allow the user to readily reuse previously defined motions. In addition, to facilitate the configuration process, we provide shortcut tools such as those for editing *Atomic Postures* (discussed in 4.1.3), frequently used postures. The user can simply combine and/or modify atomic postures for detecting sophisticated motions.

- ***Mechanisms for health monitoring***: A framework for assessment of players' heath during motion gaming has been presented [12]. Mechanisms for monitoring of unhealthy postures and measuring of the amount of body movements are to be added into UKI. The goal of this feature is to promote healthy use of motion devices and to provide an evaluation tool on health promotion researches.

- ***Automatic motion recognition***: This module of UKI aims at allowing the user to add new motions to Motion Database by only performing them. The goal of this module is to promote ease-of-use of the middleware and to make the creation of the MAP file become more intuitive. Its development has been partially completed; from our observation, the performance of recognition on a variety of motions with the low-to-moderate level of complexity is satisfactory in our pilot studies—some previews can be seen in our website [13]. However, the current version does not make full use of various types of conditions (4.1.3) yet, and we believe that, if all types are employed, the performance will be dramatically increased, and readability of generated motion models will be maximized. Therefore, we continue to conduct research on improving the performance of this module with the objective of making it become an outstanding feature of UKI. Development of this module is future work that we plan to introduce as further work.

- ***Communication over network***: Kinect2Scratch has no module for communication over network, while FAAST implements Virtual Reality Peripheral Network (VPRN) [14] for broadcasting the positions of skeleton joints to a networked application running on a server. VRPN is a network-transparent interface that provides connections between a virtual reality application and its physical devices; in order to establish a connection, both host and server applications are required to have specific functions for connecting each other. In FAAST, VRPN is necessary for using 3DVIA Studio and the Vizard virtual reality toolkit from WorldViz—they are commercial virtual world development tools that officially support FAAST.

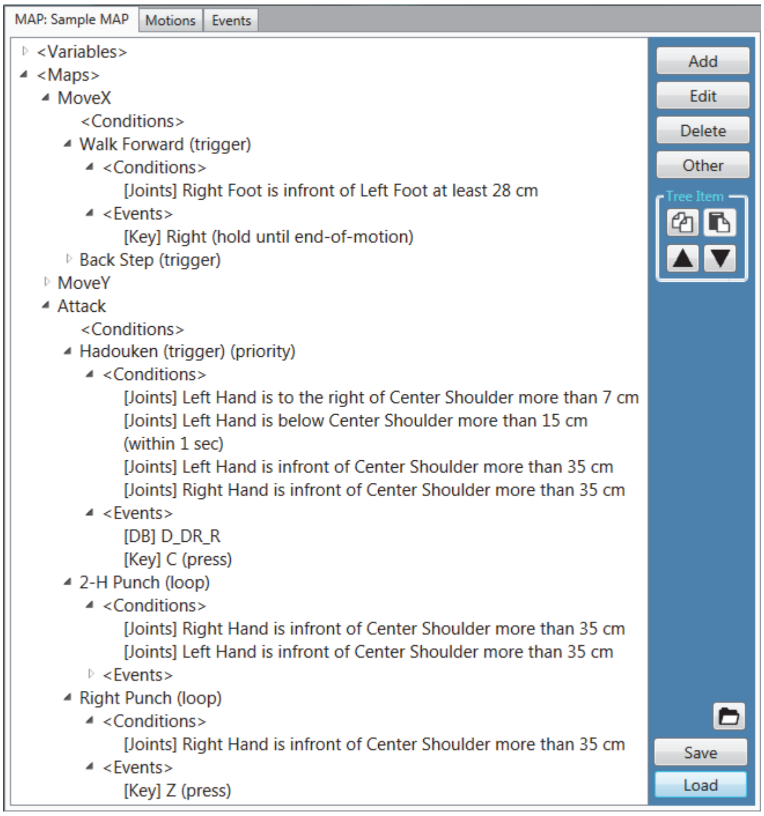
  In UKI, VRPN is not considered necessary at present because there is no networked application, in which VRPN is required, to work with. However, we have plans to implement networking functions for supporting multiplayer gaming and for sending commands from one computer to control application(s) in other computer(s). The idea is to let UKI write files containing commands (discussed in 4.1.3), or sending direct commands, over the network; the terminal applications will read commands from such files and simulate keyboard and/or mouse-clicks events. With this strategy, applications for an operating system which is not supported by the Kinect driver (e.g., Macintosh OS X), can benefit from Kinect and UKI without the need for a virtual environment.

4. DETAILED DESCRIPTIONS

In this section, we describe all components of the MAP file in detail and state how it is processed by the middleware.

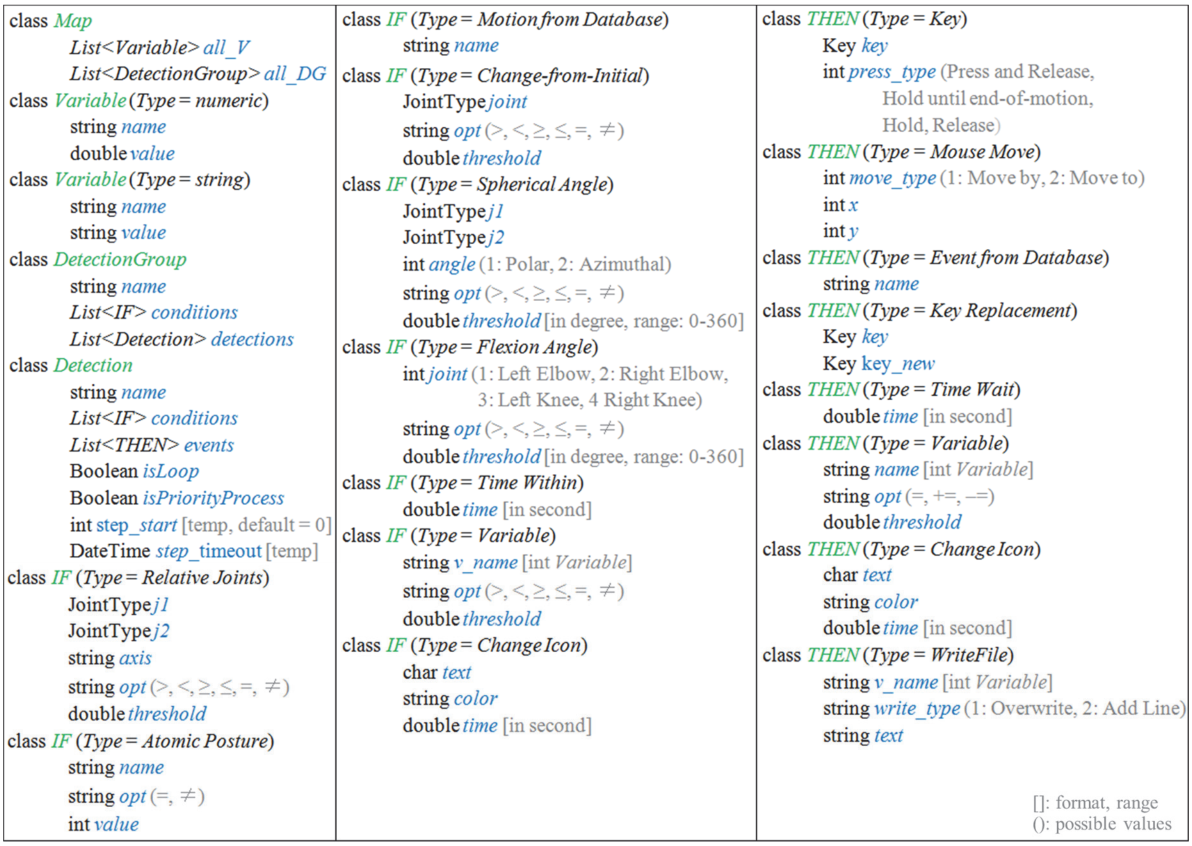
4.1. MAP-file components

The MAP file contains a list of mapping components for a specific application. MAP files are stored in an XML extension, called a storage form. A file of interest will be decoded into a human-interpretable form, called a visual form (Figure 6), during Process 3.0 (Figure 2). In addition, while being temporary stored in the middleware, the MAP file will be in code form (Figure 7); the whole MAP file is a *MAP* class instance with two parameters: *all_variables* and *all_detectionGroups*. Unless stated otherwise, please refer to Figure 7 when we describe each MAP-file component or class that consists of multiple parameters.



```
▷ <Variables>
◢ <Maps>
    ◢ MoveX
        <Conditions>
        ◢ Walk Forward (trigger)
            ◢ <Conditions>
                [Joints] Right Foot is infront of Left Foot at least 28 cm
            ◢ <Events>
                [Key] Right (hold until end-of-motion)
        ▷ Back Step (trigger)
    ▷ MoveY
    ◢ Attack
        <Conditions>
        ◢ Hadouken (trigger) (priority)
            ◢ <Conditions>
                [Joints] Left Hand is to the right of Center Shoulder more than 7 cm
                [Joints] Left Hand is below Center Shoulder more than 15 cm
                (within 1 sec)
                [Joints] Left Hand is infront of Center Shoulder more than 35 cm
                [Joints] Right Hand is infront of Center Shoulder more than 35 cm
            ◢ <Events>
                [DB] D_DR_R
                [Key] C (press)
        ◢ 2-H Punch (loop)
            ◢ <Conditions>
                [Joints] Right Hand is infront of Center Shoulder more than 35 cm
                [Joints] Left Hand is infront of Center Shoulder more than 35 cm
            ▷ <Events>
        ◢ Right Punch (loop)
            ◢ <Conditions>
                [Joints] Right Hand is infront of Center Shoulder more than 35 cm
            ◢ <Events>
                [Key] Z (press)
```

Figure 6. An example part of a Map file in its visual form.



```
class Map
        List<Variable> all_V
        List<DetectionGroup> all_DG
class Variable (Type = numeric)
        string name
        double value
class Variable (Type = string)
        string name
        string value
class DetectionGroup
        string name
        List<IF> conditions
        List<Detection> detections
class Detection
        string name
        List<IF> conditions
        List<THEN> events
        Boolean isLoop
        Boolean isPriorityProcess
        int step_start [temp, default = 0]
        DateTime step_timeout [temp]
class IF (Type = Relative Joints)
        JointType j1
        JointType j2
        string axis
        string opt (>, <, ≥, ≤, =, ≠)
        double threshold
class IF (Type = Atomic Posture)
        string name
        string opt (=, ≠)
        int value
```

```
class IF (Type = Motion from Database)
        string name
class IF (Type = Change-from-Initial)
        JointType joint
        string opt (>, <, ≥, ≤, =, ≠)
        double threshold
class IF (Type = Spherical Angle)
        JointType j1
        JointType j2
        int angle (1: Polar, 2: Azimuthal)
        string opt (>, <, ≥, ≤, =, ≠)
        double threshold [in degree, range: 0-360]
class IF (Type = Flexion Angle)
        int joint (1: Left Elbow, 2: Right Elbow,
                3: Left Knee, 4 Right Knee)
        string opt (>, <, ≥, ≤, =, ≠)
        double threshold [in degree, range: 0-360]
class IF (Type = Time Within)
        double time [in second]
class IF (Type = Variable)
        string v_name [int Variable]
        string opt (>, <, ≥, ≤, =, ≠)
        double threshold
class IF (Type = Change Icon)
        char text
        string color
        double time [in second]
```

```
class THEN (Type = Key)
        Key key
        int press_type (Press and Release,
                Hold until end-of-motion,
                Hold, Release)
class THEN (Type = Mouse Move)
        int move_type (1: Move by, 2: Move to)
        int x
        int y
class THEN (Type = Event from Database)
        string name
class THEN (Type = Key Replacement)
        Key key
        Key key_new
class THEN (Type = Time Wait)
        double time [in second]
class THEN (Type = Variable)
        string name [int Variable]
        string opt (=, +=, -=)
        double threshold
class THEN (Type = Change Icon)
        char text
        string color
        double time [in second]
class THEN (Type = WriteFile)
        string v_name [int Variable]
        string write_type (1: Overwrite, 2: Add Line)
        string text

                                        []: format, range
                                        (): possible values
```

Figure 7. All components of the MAP file in its code form.

4.1.1. DetectionGroup

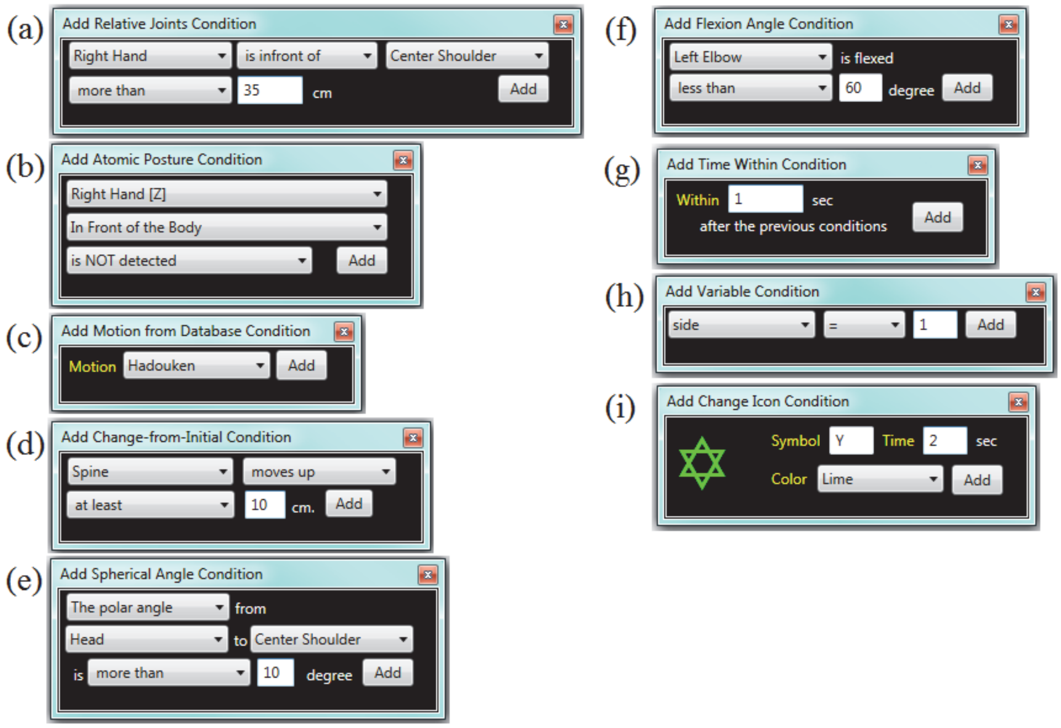
DetectionGroup is a collection of *Detection* class instances. Within a group, only one motion can be detected at a time, and the priority of detection is from top to bottom. On the other hand, the motions of different groups can be simultaneously detected and combined. For example, control configurations in 2D games are commonly divided into three different action dimensions: movements on the horizontal axis, movements on the vertical axis, and attack skills (e.g., Figure 6).

4.1.2. Detection

Detection is a component that maps conditions (*List<IF>*) to events (*List<THEN>*). Here, *List<IF>* defines a set of conditions, including those for detecting a motion (a motion model) or a series of motions (motion models), while *List<THEN>* defines a set of events, including those for specifying what to do when a motion or a series of motions is detected. For example, a concatenation of motion models for detecting “Right Punch” and “Left Punch”—with no *Time Within* condition at the concatenation point—is used for detecting “Two-Handed Punch.” On the other hand, if *Time Within* condition is inserted, the concatenation will be used for detecting “Double Punch Combo,” a 2-step punching motion. The aforementioned example indicates that a parameter *conditions* sometimes represents a combination or a sequence of motion models.

4.1.3. IF

IF is the most primitive condition for checking whether a *Detection* will be fired. Available types of *IF* (Figure 8) are as follows:



(a) Add Relative Joints Condition: Right Hand is infront of Center Shoulder; more than 35 cm

(b) Add Atomic Posture Condition: Right Hand [Z]; In Front of the Body; is NOT detected

(c) Add Motion from Database Condition: Motion Hadouken

(d) Add Change-from-Initial Condition: Spine moves up; at least 10 cm.

(e) Add Spherical Angle Condition: The polar angle from Head to Center Shoulder is more than 10 degree

(f) Add Flexion Angle Condition: Left Elbow is flexed; less than 60 degree

(g) Add Time Within Condition: Within 1 sec after the previous conditions

(h) Add Variable Condition: side = 1

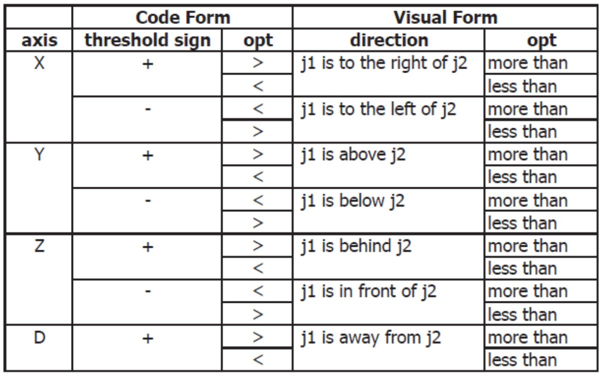
(i) Add Change Icon Condition: Symbol Y Time 2 sec; Color Lime

Figure 8. GUIs for adding a condition.

- ***Relative Joints*** (Figure 8(a)) is based on a relative difference in positions between two body joints over a given axis. Besides the common Kinect axes (*X, Y, Z*), another axis *D* is introduced by us for representing the Euclidian distance between two joints, say *j1* and *j2* , and is computed by (1). The threshold of *Relative Joints* can be negative in code form, but not in visual form due to the decoding process (Table 2) for the sake of interpretability.

$$D = \sqrt{\left(X_{j1} - X_{j2}\right)^2 + \left(Y_{j1} - Y_{j2}\right)^2 + \left(Z_{j1} - Z_{j2}\right)^2} \qquad (1)$$

Table II. Decoding table for *Relative Joints*.



Code Form			Visual Form	
axis	threshold sign	opt	direction	opt
X	+	>	j1 is to the right of j2	more than
		<		less than
	-	<	j1 is to the left of j2	more than
		>		less than
Y	+	>	j1 is above j2	more than
		<		less than
	-	<	j1 is below j2	more than
		>		less than
Z	+	>	j1 is behind j2	more than
		<		less than
	-	<	j1 is in front of j2	more than
		>		less than
D	+	>	j1 is away from j2	more than
		<		less than

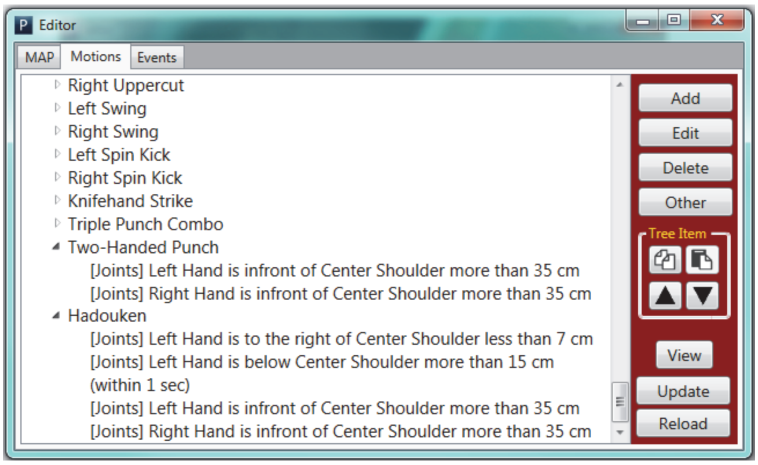
- ***Atomic Posture*** (Figure 8(b)) of a frequently used posture is provided so that users do not need to determine conditions for detecting it manually (see, e.g., “Two-Handed Punch” in Figure 9). The threshold of each *Atomic Posture* is initially given by us. It is possible to modify them by converting an *Atomic Posture* of interest into a modifiable type of condition, using the button we provide—conversion details are in the Appendix.

- ***Motion from Database*** (Figure 8(c)) is a condition for querying a stored motion model from the Motion Database. During gameplay, *IF* in this type will be replaced with *List<IF>* queried from the database by using the *name* parameter as an identity key (Figure 7 and Figure 10). Figure 11 shows how this type can be applied to the MAP file in Figure 6.

- ***Change-from-Initial*** (Figure 8(d)) detects a change in a body joint of interest on a given axis. Such a change is measured by comparing the current posture to the initial posture. *Change-from-Initial* is effective at detection of motions such as~~like~~ jumping and crouching.

- ***Spherical Angle*** (Figure 8(e)) measures an angle from a joint of interest to another in a spherical coordinate system. Such an angle can be either a polar angle (θ) or an azimuthal angle (φ) (see, e.g., Figure 12(a)). An example of its use is that of detecting of head orientations: pitch, yaw and roll.

- ***Flexion Angle*** (Figure 8(f)) measures a range of motion [15] in the joints of the body. There are four flexion angles in total, which are those of two elbows and two knees. The angles are measured between three joints by using the law of cosines in vector formulation[2] (see, e.g., Figure 12(b)).

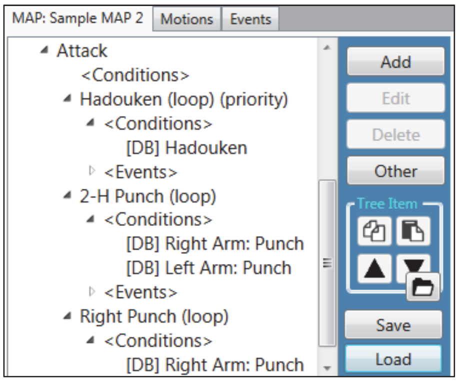


- Right Uppercut
- Left Swing
- Right Swing
- Left Spin Kick
- Right Spin Kick
- Knifehand Strike
- Triple Punch Combo
- Two-Handed Punch
  - [Joints] Left Hand is infront of Center Shoulder more than 35 cm
  - [Joints] Right Hand is infront of Center Shoulder more than 35 cm
- Hadouken
  - [Joints] Left Hand is to the right of Center Shoulder less than 7 cm
  - [Joints] Left Hand is below Center Shoulder more than 15 cm
  - (within 1 sec)
  - [Joints] Left Hand is infront of Center Shoulder more than 35 cm
  - [Joints] Right Hand is infront of Center Shoulder more than 35 cm

Figure 9. Motion models in the Motion Database.

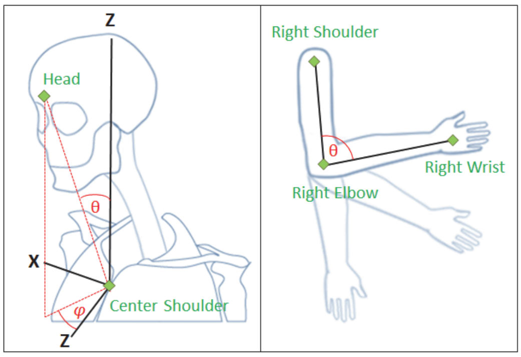
class *MotionDatabase* *List<Motion> motions_db*	class *Motion* string *name* *List<IF> conditions*	class *EventDatabase* *List<Event> events_db*	class *Event* string *name* *List<IF> events*

Figure 10. The Databases in their code form.



- Attack
  - <Conditions>
  - Hadouken (loop) (priority)
    - <Conditions>
      - [DB] Hadouken
    - <Events>
  - 2-H Punch (loop)
    - <Conditions>
      - [DB] Right Arm: Punch
      - [DB] Left Arm: Punch
    - <Events>
  - Right Punch (loop)
    - <Conditions>
      - [DB] Right Arm: Punch

Figure 11. Use of *Motion from Database* conditions in a MAP file.



(a) (b)

Figure 12. (a) Spherical angles for detecting head orientations
(b) A flexion angle of the right elbow.

- ***Time Within*** (Figure 8(g)) is used for chronologically separating conditions. The primary use is for dividing a motion into a sequence of postures.

- ***Variable*** (Figure 8(h)) is a condition based on the value of a specific *Variable* (discussed in 4.5).
- ***Change Icon*** (Figure 8(i)) is for providing visual feedback on the firing status of conditions to the user through a snowflake icon [16], as shown in Figure 1. Example usages are shown in Figure 13.

4.1.4. THEN

THEN is an event, or action to be taken when a *Detection* is fired. Available event types (Figure 14) are as follows:

- ***Key*** (Figure 14(a)) is an event for simulating a key action.
  - *Press* for pressing and subsequently releasing a key.
  - *Hold until end-of-motion* for holding a key as long as conditions in the *Detection* hold.
  - *Hold* for holding a specified key.
  - *Release* for releasing a specified key currently being pressed.

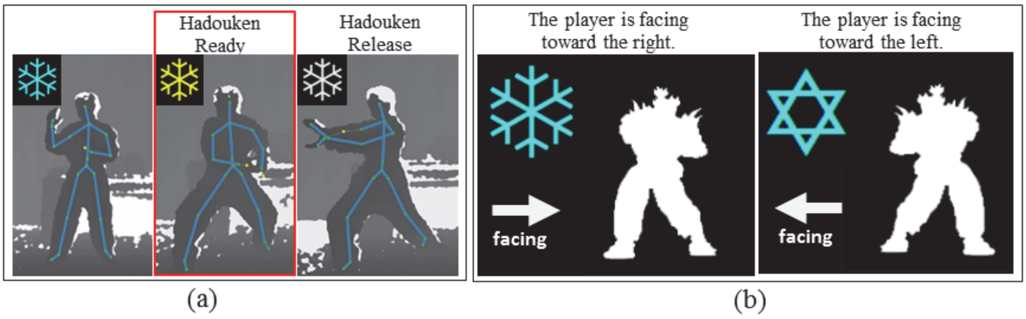
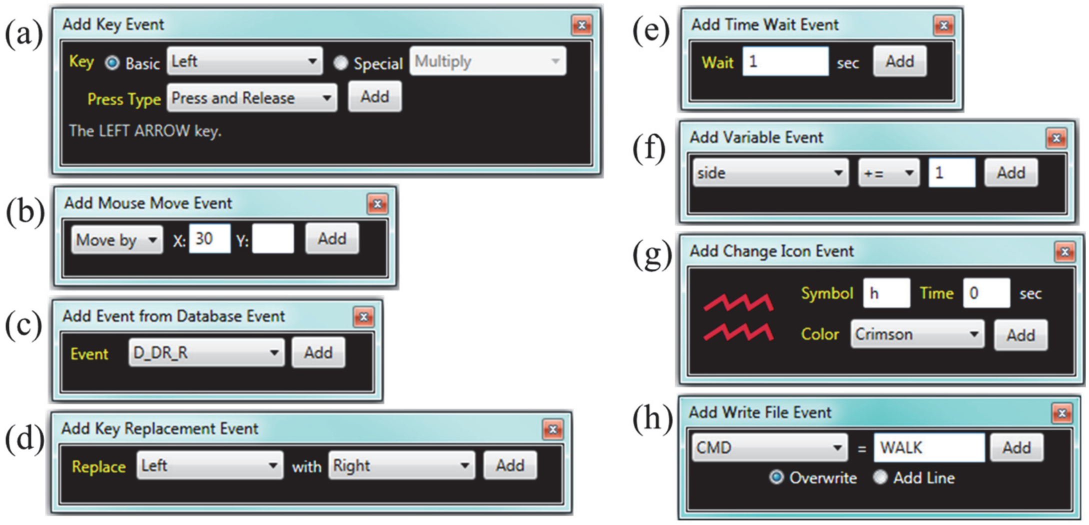


Figure 13. (a) The icon is changed after the first posture of a motion, say, Hadouken [3], is detected to indicate that the second posture is ready to be performed. (b) The icon is changed to indicate the currently active set of dynamic configurations (discussed in 4.5).



(a) Add Key Event

Key ● Basic Left ○ Special Multiply

Press Type Press and Release Add

The LEFT ARROW key.

(b) Add Mouse Move Event

Move by X: 30 Y: Add

(c) Add Event from Database Event

Event D_DR_R Add

(d) Add Key Replacement Event

Replace Left with Right Add

(e) Add Time Wait Event

Wait 1 sec Add

(f) Add Variable Event

side += 1 Add

(g) Add Change Icon Event

Symbol h Time 0 sec

Color Crimson Add

(h) Add Write File Event

CMD = WALK Add

● Overwrite ○ Add Line

Figure 14. GUIs for adding an event.

- ***Mouse Move*** (Figure 14(b)) is an event for simulating a cursor movement by either specifying the amount of movement (*Move by*) or locating an exacted position to point the cursor at (*Move to*).
- ***Event from Database*** (Figure 14(c)) is an event for querying *List<THEN>* from the Event Database.
- ***Key Replacement*** (Figure 14(d)) is an event used to shift key actions from a specified key to another. It inserts a pair of source and destination keys into a list called *Replacing List*. When a key is a source in *Replacing List*, all actions associated with the key will be applied to its destination key instead.

- ***Key Replacement*** (Figure 14(d)) is an event used to shift key actions from a specified key to another. It inserts a pair of source and destination keys into a list called *Replacing List*. When a key is a source in *Replacing List*, all actions associated with the key will be applied to its destination key instead.

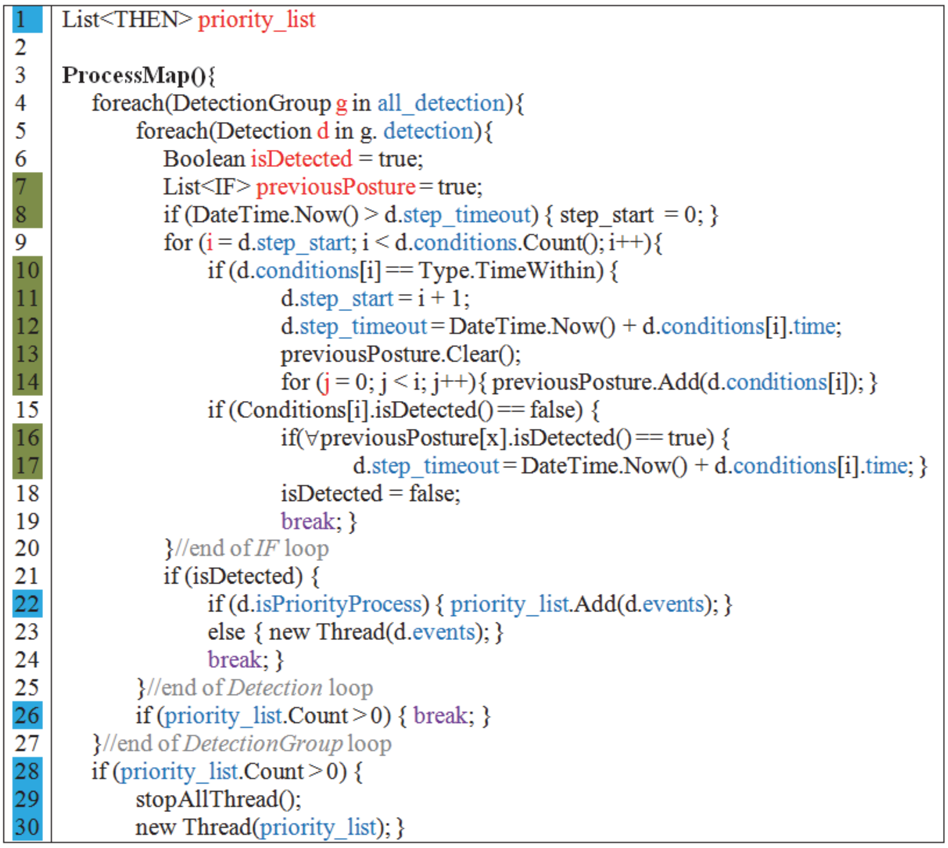
- ***Time Wait*** (Figure 14(e)) is used for separating multiple events in order to sequentially perform them; if not separated by *Time Wait*, they will be performed simultaneously.
- ***Variable*** (Figure 14(f)) is an event to specify the value of *Variable*.
- ***Change Icon*** (Figure 14(g)) has the same role as condition *Change Icon*.
- ***Write File*** (Figure 14(h)) is an event for writing a file at a path specified by a given *Variable*. In this example, a *Variable* namely “CMD” is set to “*C:\CMD.txt*”. For example of use, we have been using this event for sending direct commands from UKI running on one computer to AI in a fighting game running on another computer in a network [13]—it helps to enable multiplayer gameplay, and no decrease in performance has been seen on LAN connection. This event can be used to establish communication over network, by adding a small application that reads the written file and simulates keyboard/mouse events based on commands in the read file.

4.2. Motion detection based on the MAP file

In our terms, “Posture” is a snapshot of the skeleton at a point of time. On the other hand, “Motion” is a sequence of postures over time. Motions that consist of only one posture are called single-step whereas the others are called multiple-step or *n*-step. Figure 15 shows the pseudocode for detecting motions based on the MAP file.

4.2.1. Detection of single-step motions

From the pseudocode, lines without background color in their numbers are for detecting single-step motions. As there is only one stage in such motions, mechanisms for monitoring the stage of motion are not required. Once all conditions in *List<IF>* of a *Detection* are satisfied, the events in *List<THEN>* for that *Detection* will be processed by running a thread (Figure 16).



```
List<THEN> priority_list

ProcessMap(){
  foreach(DetectionGroup g in all_detection){
      foreach(Detection d in g. detection){
          Boolean isDetected = true;
          List<IF> previousPosture = true;
          if (DateTime.Now() > d.step_timeout) { step_start = 0; }
          for (i = d.step_start; i < d.conditions.Count(); i++){
              if (d.conditions[i] == Type.TimeWithin) {
                      d.step_start = i + 1;
                      d.step_timeout = DateTime.Now() + d.conditions[i].time;
                      previousPosture.Clear();
                      for (j = 0; j < i; j++){ previousPosture.Add(d.conditions[i]); }
              if (Conditions[i].isDetected() == false) {
                      if(∀previousPosture[x].isDetected() == true) {
                              d.step_timeout = DateTime.Now() + d.conditions[i].time; }
                      isDetected = false;
                      break; }
          }//end of IF loop
          if (isDetected) {
              if (d.isPriorityProcess) { priority_list.Add(d.events); }
              else { new Thread(d.events); }
              break; }
      }//end of Detection loop
      if (priority_list.Count > 0) { break; }
  }//end of DetectionGroup loop
  if (priority_list.Count > 0) {
      stopAllThread();
      new Thread(priority_list); }
```

Figure 15. Pseudocode for detecting motions based on the MAP file.

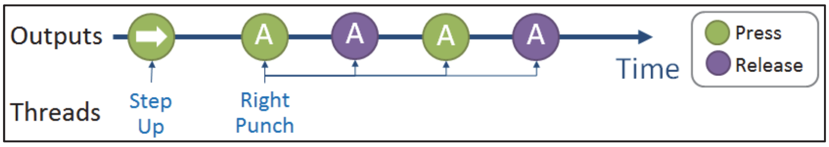


Figure 16. A combination of outputs by threads.

4.2.2. Detection of multiple-step motions

Because it is impossible to detect all postures in a multiple-step motion at once, mechanisms for monitoring the stage of a motion become necessary. Therefore, parameters for storing the transition stage are introduced (*step_start* and *step_timeout* in Figure 7), and lines with green background on their numbers are added to *ProcessMap*() (Figure 15).

Suppose the MAP file in Figure 6 is used and at time *t*, the player is performing the first motion of "Hadouken." Among its five conditions, the first two conditions are satisfied, and hence the detection checking process (DCP) reaches the *Time Within* condition, by which details on the current stage are kept in *step_start* and *step_timeout*. "Hadouken" is not yet fired at time *t* because the fourth and fifth conditions are not satisfied. However, on further *ProcessMap*() iterations, when DCP enters "Hadouken," it will start checking conditions from the fourth condition (the first condition of the second posture). Thereby, if the second posture is detected within one second, counted from EoM of the first motion, detection of the whole "Hadouken" will be completed.

4.3. "Loop" and "Priority Process"

The *Detection* has two optional properties that can be activated or deactivated as follows:

- ***Loop*** is the property for repeatedly detecting motions and sending outputs over time. For example, according to the MAP file in Figure 6, if the player does and holds "Right Punch," the middleware will continuously send commands for pressing and releasing key Z over time. On the other hand, if the *Loop* property is deactivated, key Z will be pressed and released only once.

- ***PriorityProcess*** is the property that ensures *events* of the *Detection* will be processed without being interrupted. Lines with blue background on their numbers are added to *ProcessMap*() (Figure 15). For example, suppose during performing "Step Forward," the player does "Hadouken." If "Hadouken" is defined as a priority process, its *events* will not be processed by running a normal thread. Instead, it will be put into a list called *priority_list*. Once *priority_list* becomes non-empty, DCP will terminate, all running threads will be forced to stop, and all keys that have been pressed will be released. Items in *priority_list* will then be processed. Interruptions are prevented until this processing is finished (e.g., Figure 17).

4.4. Use of Variables

A *Variable* can be used as a switch or a temporary memory for keeping a numeric value during gameplay.

- ***Changing the facing direction***: *Variables* can be applied for solving the facing direction problem (3.2.4). An example is shown in Figure 18: when the player changes the character's facing direction and raises the left hand up, left and right will be reversed (1). To reverse back, the left hand must be lowered down below the spine height (3) before raising it up again (2).

- ***Changing game controls dynamically***: There are games in which the player can change weapons or skill sets, or switching characters during gameplay. An example on handling character switching in *Street Fighter X Tekken* [17] is shown in Figure 19. The concept is similar to the facing direction problem; the value of *cha* will be switched between 1 and 2 when a specified motion is performed. This *Variable* "cha" is used as a switch for turning on or off *DetectionGroups* to match the character in use.

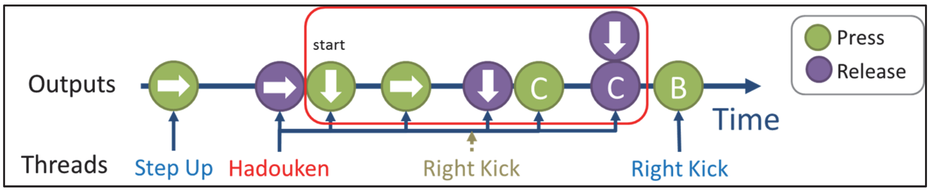
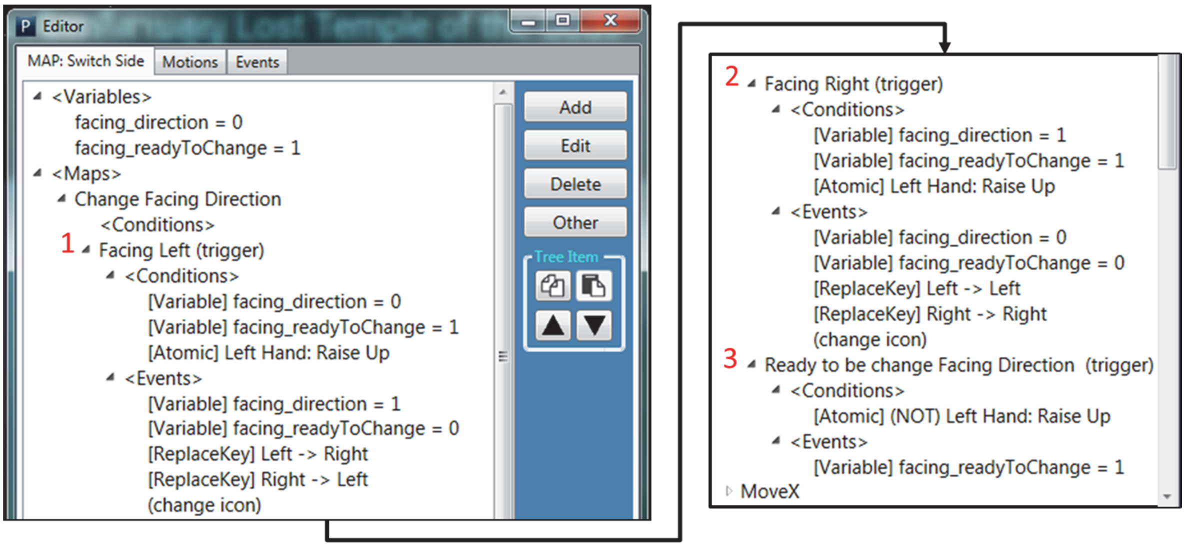


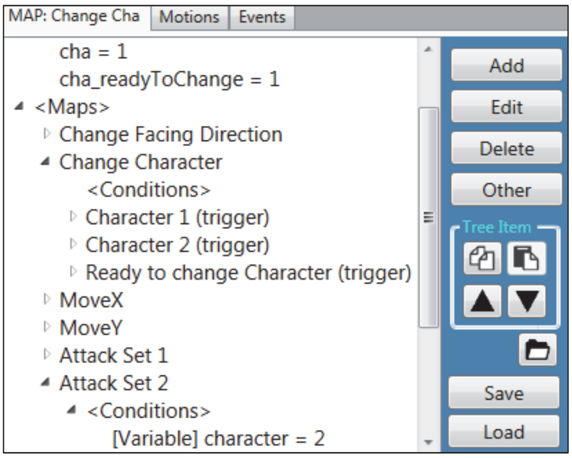
Figure 17. Time line for processing of the Hadouken button combo with *PriorityProcess*.



```
◢ <Variables>
    facing_direction = 0
    facing_readyToChange = 1
◢ <Maps>
  ◢ Change Facing Direction
      <Conditions>
    ◢ Facing Left (trigger)
      ◢ <Conditions>
          [Variable] facing_direction = 0
          [Variable] facing_readyToChange = 1
          [Atomic] Left Hand: Raise Up
      ◢ <Events>
          [Variable] facing_direction = 1
          [Variable] facing_readyToChange = 0
          [ReplaceKey] Left -> Right
          [ReplaceKey] Right -> Left
          (change icon)
```

```
◢ Facing Right (trigger)
  ◢ <Conditions>
      [Variable] facing_direction = 1
      [Variable] facing_readyToChange = 1
      [Atomic] Left Hand: Raise Up
  ◢ <Events>
      [Variable] facing_direction = 0
      [Variable] facing_readyToChange = 0
      [ReplaceKey] Left -> Left
      [ReplaceKey] Right -> Right
      (change icon)
◢ Ready to be change Facing Direction  (trigger)
  ◢ <Conditions>
      [Atomic] (NOT) Left Hand: Raise Up
  ◢ <Events>
      [Variable] facing_readyToChange = 1
▷ MoveX
```

Figure 18. Handling of changing facing direction in 2D gamesm by adding *DetectionGroup* called "Change Facing Direction" with three *Detection*s and two *Variable*s. *facing_direction* is for indicating the player character's facing direction, while *facing_readyToChange* is for checking whether the facing direction is ready to be changed.

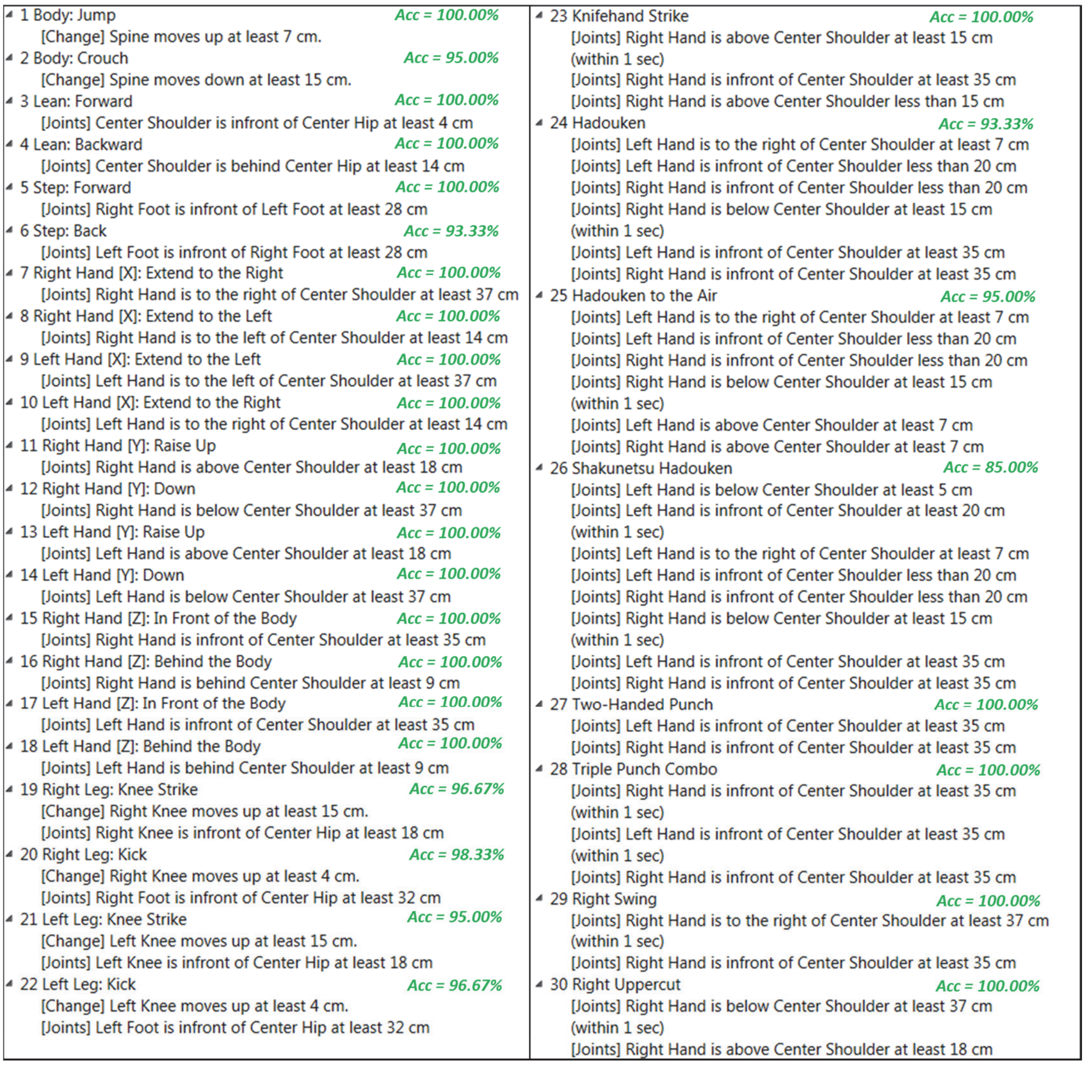


```
    cha = 1
    cha_readyToChange = 1
◢ <Maps>
  ▷ Change Facing Direction
  ◢ Change Character
      <Conditions>
    ▷ Character 1 (trigger)
    ▷ Character 2 (trigger)
    ▷ Ready to change Character (trigger)
  ▷ MoveX
  ▷ MoveY
  ▷ Attack Set 1
  ◢ Attack Set 2
    ◢ <Conditions>
        [Variable] character = 2
```

Figure 19. Handling of character switching.

5. PERFORMANCE

An experiment was conducted to evaluate the performance of UKI on 30 sample motions. For each motion, three data instances were collected from each subject out of 20 subjects involved. For each motion, an optimal model was built according to an analysis approach [3]: the motion analyst selects candidate features, initially recommended by a D-Tree classifier, and empirically determines their threshold values based on exploratory data analysis. The detection accuracy of a motion is represented by the probability that the motion of interest is detected on its 60 instances.



1 Body: Jump Acc = 100.00%
[Change] Spine moves up at least 7 cm.
2 Body: Crouch Acc = 95.00%
[Change] Spine moves down at least 15 cm.
3 Lean: Forward Acc = 100.00%
[Joints] Center Shoulder is infront of Center Hip at least 4 cm
4 Lean: Backward Acc = 100.00%
[Joints] Center Shoulder is behind Center Hip at least 14 cm
5 Step: Forward Acc = 100.00%
[Joints] Right Foot is infront of Left Foot at least 28 cm
6 Step: Back Acc = 93.33%
[Joints] Left Foot is infront of Right Foot at least 28 cm
7 Right Hand [X]: Extend to the Right Acc = 100.00%
[Joints] Right Hand is to the right of Center Shoulder at least 37 cm
8 Right Hand [X]: Extend to the Left Acc = 100.00%
[Joints] Right Hand is to the left of Center Shoulder at least 14 cm
9 Left Hand [X]: Extend to the Left Acc = 100.00%
[Joints] Left Hand is to the left of Center Shoulder at least 37 cm
10 Left Hand [X]: Extend to the Right Acc = 100.00%
[Joints] Left Hand is to the right of Center Shoulder at least 14 cm
11 Right Hand [Y]: Raise Up Acc = 100.00%
[Joints] Right Hand is above Center Shoulder at least 18 cm
12 Right Hand [Y]: Down Acc = 100.00%
[Joints] Right Hand is below Center Shoulder at least 37 cm
13 Left Hand [Y]: Raise Up Acc = 100.00%
[Joints] Left Hand is above Center Shoulder at least 18 cm
14 Left Hand [Y]: Down Acc = 100.00%
[Joints] Left Hand is below Center Shoulder at least 37 cm
15 Right Hand [Z]: In Front of the Body Acc = 100.00%
[Joints] Right Hand is infront of Center Shoulder at least 35 cm
16 Right Hand [Z]: Behind the Body Acc = 100.00%
[Joints] Right Hand is behind Center Shoulder at least 9 cm
17 Left Hand [Z]: In Front of the Body Acc = 100.00%
[Joints] Left Hand is infront of Center Shoulder at least 35 cm
18 Left Hand [Z]: Behind the Body Acc = 100.00%
[Joints] Left Hand is behind Center Shoulder at least 9 cm
19 Right Leg: Knee Strike Acc = 96.67%
[Change] Right Knee moves up at least 15 cm.
[Joints] Right Knee is infront of Center Hip at least 18 cm
20 Right Leg: Kick Acc = 98.33%
[Change] Right Knee moves up at least 4 cm.
[Joints] Right Foot is infront of Center Hip at least 32 cm
21 Left Leg: Knee Strike Acc = 95.00%
[Change] Left Knee moves up at least 15 cm.
[Joints] Left Knee is infront of Center Hip at least 18 cm
22 Left Leg: Kick Acc = 96.67%
[Change] Left Knee moves up at least 4 cm.
[Joints] Left Foot is infront of Center Hip at least 32 cm
23 Knifehand Strike Acc = 100.00%
[Joints] Right Hand is above Center Shoulder at least 15 cm
(within 1 sec)
[Joints] Right Hand is infront of Center Shoulder at least 35 cm
[Joints] Right Hand is above Center Shoulder less than 15 cm
24 Hadouken Acc = 93.33%
[Joints] Left Hand is to the right of Center Shoulder at least 7 cm
[Joints] Left Hand is infront of Center Shoulder less than 20 cm
[Joints] Right Hand is infront of Center Shoulder less than 20 cm
[Joints] Right Hand is below Center Shoulder at least 15 cm
(within 1 sec)
[Joints] Left Hand is infront of Center Shoulder at least 35 cm
[Joints] Right Hand is infront of Center Shoulder at least 35 cm
25 Hadouken to the Air Acc = 95.00%
[Joints] Left Hand is to the right of Center Shoulder at least 7 cm
[Joints] Left Hand is infront of Center Shoulder less than 20 cm
[Joints] Right Hand is infront of Center Shoulder less than 20 cm
[Joints] Right Hand is below Center Shoulder at least 15 cm
(within 1 sec)
[Joints] Left Hand is above Center Shoulder at least 7 cm
[Joints] Right Hand is above Center Shoulder at least 7 cm
26 Shakunetsu Hadouken Acc = 85.00%
[Joints] Left Hand is below Center Shoulder at least 5 cm
[Joints] Left Hand is infront of Center Shoulder at least 20 cm
(within 1 sec)
[Joints] Left Hand is to the right of Center Shoulder at least 7 cm
[Joints] Left Hand is infront of Center Shoulder less than 20 cm
[Joints] Right Hand is infront of Center Shoulder less than 20 cm
[Joints] Right Hand is below Center Shoulder at least 15 cm
(within 1 sec)
[Joints] Left Hand is infront of Center Shoulder at least 35 cm
[Joints] Right Hand is infront of Center Shoulder at least 35 cm
27 Two-Handed Punch Acc = 100.00%
[Joints] Left Hand is infront of Center Shoulder at least 35 cm
[Joints] Right Hand is infront of Center Shoulder at least 35 cm
28 Triple Punch Combo Acc = 100.00%
[Joints] Right Hand is infront of Center Shoulder at least 35 cm
(within 1 sec)
[Joints] Left Hand is infront of Center Shoulder at least 35 cm
(within 1 sec)
[Joints] Right Hand is infront of Center Shoulder at least 35 cm
29 Right Swing Acc = 100.00%
[Joints] Right Hand is to the right of Center Shoulder at least 37 cm
(within 1 sec)
[Joints] Right Hand is infront of Center Shoulder at least 35 cm
30 Right Uppercut Acc = 100.00%
[Joints] Right Hand is below Center Shoulder at least 37 cm
(within 1 sec)
[Joints] Right Hand is above Center Shoulder at least 18 cm

Figure 20. Optimal models of the tested motions.

The list of all motions, including their optimal models and detection accuracies, is shown in Figure 20. The average detection accuracies are 98.3% for all motions. These results indicate that the modeling method available in UKI effectively represents common patterns in motion data—a model of each posture not only consists of only a few features, all interpretable to their analysts or users, but also can classify 60 instances at a high rate of accuracy.

6. IMPACT AND USES

The middleware can be used for many purposes, including general gaming and research. Some sample uses and potential benefits of UKI are given in this section.

6.1. General gameplay

In this subsection, we recommend genres of games suitable for playing with Kinect. Examples are from notable games where official Kinect versions are not available. In addition, our collections of demo videos [13] are available in our UKI website.

- ***2D games***: It has been proven in our previous studies [3, 4] that UKI is capable of controlling the character in fighting games, as effectively as when using a keyboard or joystick. We consider that fighting games (e.g., *Street Fighter* [18], *Tekken* [19], and *FightingICE* [20]) are perfectly suitable to be played with Kinect as exer-games. Other suitable genres are, for examples, beat 'em up (e.g., Kung-Fu Master [21]), hack-and-slash (e.g., Achilles [22]), dodging-and-jumping (e.g., Donkey Kong [23]) and MMORPG (e.g., Maple Story Online [24]).
- ***3D games***: One recommended genre is hack-and-slash where a combat with hand-to-hand weapons is focused (e.g., *Dynasty Warriors* and *Samurai Warriors* series by Koei [25]). Another recommended genre is the walking simulator; for example, *Be Lost* [26], a survival horror game that supports Oculus Rift. First-person shooting games, such as~~like~~ *Counter Strike* and *Call of Duty* [27], are optional, where a challenge lies in how to control the viewpoint and crosshair—an additional device such as~~like~~ GunCon/G-Con [28] may be needed.

We believe UKI as an alternative game controller will provide gamers with a unique and impressive experience—as confirmed by Mizobata et al. [29] that full-body game interaction can make gamers enjoy games more, regardless of kinds of players. Mizobata et al. also revealed that the design of motions used for full-body game interaction is one significant factor affecting player engagement; in their study, some players complained about unrealistic interaction in certain games. Motions designed by developers are not always preferable and suitable motions should be investigated [30]. However, such designs are also dependent on user preferences. UKI can solve these problems as it allows users to design their own motions for game control. It is also noted that experiments on designing motions for intense gameplay (e.g., [30]) will benefit from UKI—researchers can use UKI to test designed motions on actual gameplay without any need for Kinect programming.

6.2. Serious games & simulations

One notable application of serious games is a training system for providing instructions or simulating experiments to trainees through a game-like or virtual reality application [31]. One example is a ballet dance training system [32]; UKI can be applied to existing dancing games, and the icon in the UKI dialog (4.1.3) can be used to provide feedback whether the user correctly performs their expected motions. Other types of training applications are a medical training system, for example, a virtual reality by Mathur [33] for helping students understand human anatomy in 3D environment (Oculus Rift and Razer Hydra are currently used), and a fire drill training system [34] for learning how to evacuate in the event of a fire (keyboard and mouse are currently used). While the aforementioned medical training system may use UKI for simulating anatomy using hand controls, the aforementioned fire drill training system has already been using UKI [35].

6.3. Games for health & medical application

Motion gaming was reported to be an effective means for treatment of chronic disease [36], increasing physical activity (PA) [37, 38], and improving physiologic responses [39]. In the medical field, Kinect is widely accepted as a tool for developing motion games to provide functional training [40], treatment [41], and rehabilitation [42, 43, 44, 45]. These kinds of applications are collectively called games for heath (G4H), which have emerged as promising to achieve sustained therapy practice and patient motivation [36]. Both in-hospital and at-home exercises can be provided in an interactive, supervised, and motivated manner [44].

UKI can be applied for assessing and monitoring users' health. It is reported by Plantard et al. [46] that the performance of Kinect can correctly complete ergonomic assessment grids. A previous version of UKI was also used to monitor sitting postures of an office worker and to warn them when an unhealthy posture was detected [47]—this is an example of a routine health monitoring system where Kinect outperforms alternative technologies. Besides routine health monitoring, another well-known type of monitoring system is monitoring of unexpected events in daily life, such as a fall monitoring system created by Stone et al. [48]. They revealed that the performance of a Kinect-based system is equivalent to existing systems. However, the Kinect-based system was not affected by lighting conditions and could significantly reduce the cost. The availability of UKI will allow everyone to employ the described benefits of Kinect, and once the automatic motion recognition module (3.2.5) is finished, addition of new motion will be a simply task.

Health monitoring can be applied on G4H. It is mentioned that one needed theme of research is to investigate adverse outcomes from G4H [36]. An important point which many G4H researchers overlook—or avoid mentioning is that being engaged in exercise incurs the risk of sustaining injuries [49]. In responding to this problem, we introduced a project to develop health monitoring modules on UKI (previously discussed in 3.2.5). Another needed tool is that for monitoring the long-term outcome of G4H [36]. This need can be fulfilled by adding modules for collecting data during motion gameplay. Data will be processed, and health reports will be generated automatically. These reports will serve in quantitative evaluation of both positive and negative outcomes, as well as behavior changes. In case G4H is offered to patients or elders, a module for monitoring of unexpected events may also be added—for sounding the alarm to the caregiver in cases of accidents or sudden illness.

6.4. Virtual Reality and Arts Exhibition

Virtual reality (VR) and exhibition of Arts are other applications for UKI. As there are many virtual online museums that offer free tours through their collections of art, history or science[1], it is possible to apply motion control technology for creating new and unique experiences to visitors [50]. UKI, along with Oculus Rift, can be used for controlling an avatar to provide realistic virtual field trips on VR applications such as~~like~~ museum [51], laboratory [52], and time travel [53]. Another example is to use the middleware for creating touch-enabled interfaces for an interactive art and exhibition such as~~like~~ smARTbox [54]. With UKI, it is possible for businesses and artists to add motion control interfaces to their products and works, without the need to hire programmers.

7. CONCLUSION AND FUTURE WORK

We have presented UKI, a universal middleware that covers control specifications in most existing games and is adaptable to a variety of applications. The needs and reasons underlying the development of UKI were discussed. This paper focuses on describing designs of components and technical details on how UKI works. We also provided ideas and examples on existing uses of UKI. The middleware is available for use by anyone, not just general gamers or researchers, and regardless of status as novices or experts. On the official website of UKI [13], we provide sample MAP files and instructions on how to getting started with UKI as well as on advanced uses of UKI. Sample motion models from our analysis are also available for download, and data used in performance evaluation in our paper are available in the public-domain.

Our future work will focus on developments of extension features as shown in Table I; these include mechanisms for health monitoring, functions for automatic motion recognition, and networking functions. We will continue to promote uses of UKI, especially in G4H researches. We will continuously improve the middleware to make it accepted as a professional motion controlling middleware.

ACKNOWLEDGEMENT

The authors wish to thank members in Intelligent Computer Entertainment Lab, Ritsumeikan University, for their experimental helps and suggestions. We would also like to express special thanks to Mr. Makoto Ishihara, for recruiting subjects and being an assistant in data collection. Thanks also go to Samuel Henz, a former visiting guest student from EPITECH, France, for his helps in debugging the middleware.

REFERENCES

1. A Kinect-less Xbox One Signals the Death of Motion Gaming | Hardcore Gamer. http://www.hardcoregamer.com/2014/05/19/a-kinect-less-xbox-one-signals-the-death-of-motion-gaming. Accessed: 2016-11-01.
2. Microsoft Still Has Faith In The Kinect. http://www.cinemablend.com/games/Microsoft-Still-Has-Faith-Kinect-71309.html. Accessed: 2016-11-01.
3. Paliyawan P, Sookhanaphibarn K, Choensawat W, Thawonmas R. Body motion design and analysis for fighting game interface. Computational Intelligence and Games (CIG), 2015 IEEE Conference on 2015 Aug 31 (pp. 360-367). IEEE. DOI: 10.1109/CIG.2015.7317960.
4. Paliyawan P, Sookhanaphibarn K, Choensawat W, Thawonmas R. Towards universal kinect interface for fighting games. 2015 IEEE 4th Global Conference on Consumer Electronics (GCCE) 2015 Oct 27 (pp. 332-333). IEEE. DOI: 10.1109/GCCE.2015.7398599.
5. Suma EA, Krum DM, Lange B, Koenig S, Rizzo A, Bolas M. Adapting user interfaces for gestural interaction with the flexible action and articulated skeleton toolkit. Computers & Graphics. 2013 May 31;37(3):193-201. DOI:10.1016/j.cag.2012.11.004.
6. Kinect2Scratch by Stephen Howell. http://scratch.saorog.com. Accessed: 2016-11-01.
7. Ultra Street Fighter IV. http://games.streetfighter.com/us/usfiv. Accessed: 2016-11-01.
8. Kinect for Windows SDK v1.8 from Official Microsoft Download Center. https://www.microsoft.com/en-us/download/details.aspx?id=40278. Accessed: 2016-11-01.
9. Windows 10 growth sluggish as Windows 7/8.x users stick with their OS | ZDNet. http://www.zdnet.com/article/windows-10-growth-sluggish-as-windows-7-windows-8-users-stick-with-their-os. Accessed: 2016-11-01.

10. Pazhoumand-Dar H, Lam CP, Masek M. Joint movement similarities for robust 3D action recognition using skeletal data. Journal of Visual Communication and Image Representation. 2015 Jul 31;30:10-21. DOI:10.1016/j.jvcir.2015.03.002.
11. Hadoken | Street Fighter Wiki. http://streetfighter.wikia.com/wiki/Hadoken. Accessed: 2016-11-01.
12. Paliyawan P, Sookhanaphibarn K, Choensawat W, Thawonmas R. Towards Ergonomic Exergaming. 2016 IEEE 5th Global Conference on Consumer Electronics (GCCE) 2016 Oct 12 (pp. 321-323). IEEE.
13. UKI. https://sites.google.com/site/icelabuki. Accessed: 2016-11-01.
14. Taylor II RM, Hudson TC, Seeger A, Weber H, Juliano J, Helser AT. VRPN: a device-independent, network-transparent VR peripheral system. Proceedings of the ACM symposium on Virtual reality software and technology 2001 Nov 15 (pp. 55-61). ACM. DOI: 10.1145/505008.505019.
15. Range of Motion Exercises. http://alsworldwide.org/assets/misc/RANGE_OF_MOTION_EXERCISES_WITH_PHOTOS_copy.pdf. Accessed: 2016-11-01.
16. Wingdings - Wikipedia. https://en.wikipedia.org/wiki/Wingdings. Accessed: 2016-11-01.
17. Street Fighter X Tekken. http://games.streetfighter.com//sfxtk. Accessed: 2016-11-01.
18. Street Fighter. https://en.wikipedia.org/wiki/Street_Fighter. Accessed: 2016-11-01.
19. Tekken. https://en.wikipedia.org/wiki/Tekken. Accessed: 2016-11-01.
20. Welcome to Fighting Game AI Competition. http://www.ice.ci.ritsumei.ac.jp/~ftgaic. Accessed: 2016-11-01.
21. Kung-Fu Master. https://en.wikipedia.org/wiki/Kung-Fu_Master. Accessed: 2016-11-01.
22. Achilles - Ben Olding Games. http://www.benoldinggames.co.uk/games/Achilles.htm. Accessed: 2016-11-01.
23. Donkey Kong. https://en.wikipedia.org/wiki/Donkey_Kong. Accessed: 2016-11-01.
24. MapleStory. https://en.wikipedia.org/wiki/MapleStory. Accessed: 2016-11-01.
25. Koei | Koei Wiki. http://koei.wikia.com/wiki/Koei. Accessed: 2016-11-01.
26. The Hospital Haunted Be Lost. https://share.oculus.com/app/the-hospital-haunted-be-lost. Accessed: 2016-11-01.
27. Call of Duty. https://en.wikipedia.org/wiki/Call_of_Duty. Accessed: 2016-11-01.
28. GunCon. https://en.wikipedia.org/wiki/GunCon. Accessed: 2016-11-01.
29. Mizobata R, Silpasuwanchai C, Ren X. Only for casual players?: investigating player differences in full-body game interaction. Proceedings of the Second International Symposium of Chinese CHI 2014 Apr 26 (pp. 57-65). ACM. DOI: 10.1145/2592235.2592244.
30. Silpasuwanchai C, Ren X. Designing concurrent full-body gestures for intense gameplay. International Journal of Human-Computer Studies. 2015 Aug 31;80:1-3. DOI:10.1016/j.ijhcs.2015.02.010.
31. Pañella OG. Game Design and e-Health: Serious Games put to the test. Advancing Cancer Education and Healthy Living in Our Communities: Putting Visions and Innovations Into Action. Selected Papers from the St. Jude Cure4Kids® Global Summit 2011. 2012 Aug 24;172:71. DOI: 10.3233/978-1-61499-088-8-71.
32. Kyan M, Sun G, Li H, Zhong L, Muneesawang P, Dong N, Elder B, Guan L. An approach to ballet dance training through ms kinect and visualization in a cave virtual reality environment. ACM Transactions on Intelligent Systems and Technology (TIST). 2015 Mar 31;6(2):23. DOI: 10.1145/2735951.
33. Mathur AS. Low cost virtual reality for medical training. Virtual Reality (VR), 2015 IEEE 2015 Mar 23 (pp. 345-346). IEEE. DOI: 10.1109/VR.2015.7223437.
34. Sookhanaphibarn K, Choensawat W, Paliyawan P, Thawonmas R. Virtual Reality of Fire Evacuation Training in 3D Virtual World. 2016 IEEE 4th Global Conference on Consumer Electronics (GCCE) 2016 Oct 12 (pp. 323-325). IEEE.
35. Serious Game, Fire Evacuation. https://www.youtube.com/watch?v=PyvfOfvJV6M. Accessed: 2016-11-01.
36. Baranowski T, Blumberg F, Buday R, DeSmet A, Fiellin LE, Green CS, Kato PM, Lu AS, Maloney AE, Mellecker R, Morrill BA. Games for health for children—Current status and needed research. Games for health journal. 2016 Feb 1;5(1):1-2. DOI:10.1089/g4h.2015.0026.
37. Peng W, Crouse JC, Lin JH. Using active video games for physical activity promotion: a systematic review of the current state of research. Health education & behavior. 2012 Jul 6. DOI: 10.1177/1090198112444956.
38. Leutwyler H, Hubbard EM, Vinogradov S, Dowling GA. Videogames to promote physical activity in older adults with schizophrenia. Games for health journal. 2012 Oct 1;1(5):381-3. DOI: 10.1089/g4h.2012.0051.
39. Smallwood SR, Morris MM, Fallows SJ, Buckley JP. Physiologic responses and energy expenditure of Kinect active video game play in schoolchildren. Archives of pediatrics & adolescent medicine. 2012 Nov 1;166(11):1005-9. DOI:10.1001/archpediatrics.2012.1271.
40. Sato K, Kuroki K, Saiki S, Nagatomi R. Improving walking, muscle strength, and balance in the elderly with an exergame using Kinect: A randomized controlled trial. Games for health journal. 2015 Jun 1;4(3):161-7. DOI:10.1089/g4h.2014.0057.
41. Galna B, Jackson D, Schofield G, McNaney R, Webster M, Barry G, Mhiripiri D, Balaam M, Olivier P, Rochester L. Retraining function in people with Parkinson's disease using the Microsoft kinect: game design and pilot testing. Journal of neuroengineering and rehabilitation. 2014 Apr 14;11(1):1. DOI: 10.1186/1743-0003-11-60.
42. Pastor I, Hayes HA, Bamberg SJ. A feasibility study of an upper limb rehabilitation system using kinect and computer games. Engineering in Medicine and Biology Society (EMBC), 2012 Annual International Conference of the EEE 2012 Aug 28 (pp. 1286-1289). IEEE. DOI: 10.1109/EMBC.2012.6346173.
43. Saini S, Rambli DR, Sulaiman S, Zakaria MN, Shukri SR. A low-cost game framework for a home-based stroke rehabilitation system. Computer & Information Science (ICCIS), 2012 International Conference on 2012 Jun 12 (Vol. 1, pp. 55-60). IEEE. DOI: 10.1109/ICCISci.2012.6297212.
44. Da Gama A, Fallavollita P, Teichrieb V, Navab N. Motor rehabilitation using Kinect: A systematic review. Games for health journal. 2015 Apr 1;4(2):123-35. DOI:10.1089/g4h.2014.0047.
45. Da Gama A, Chaves T, Figueiredo L, Teichrieb V. Poster: improving motor rehabilitation process through a natural interaction based system using kinect sensor. In3D User Interfaces (3DUI), 2012 IEEE Symposium on 2012 Mar 4 (pp. 145-146). IEEE. DOI: 10.1109/3DUI.2012.6184203.
46. Plantard P, Auvinet E, Pierres AS, Multon F. Pose estimation with a kinect for ergonomic studies: Evaluation of the accuracy using a virtual mannequin. Sensors. 2015 Jan 15;15(1):1785-803. DOI: 10.3390/s150101785.
47. Paliyawan P, Nukoolkit C, Mongkolnam P. Office workers syndrome monitoring using kinect. Communications (APCC), 2014 Asia-Pacific Conference on 2014 Oct 1 (pp. 58-63). IEEE. DOI: 10.1109/APCC.2014.7091605.
48. Stone EE, Skubic M. Evaluation of an inexpensive depth camera for passive in-home fall risk assessment. InPervasive Computing Technologies for Healthcare (PervasiveHealth), 2011 5th International Conference on 2011 May 23 (pp. 71-77). IEEE. DOI: 10.3233/AIS-2011-0124.

49. R. Rössler et al., "Exercise-based injury prevention in child and adolescent sport: a systematic review and meta-analysis," Sports Medicine, 2014, 44(12), pp.1733-1748. DOI:10.1007/s40279-014-0234-2.

50. Jung T, tom Dieck MC, Lee H, Chung N. Effects of Virtual Reality and Augmented Reality on Visitor Experiences in Museum. Information and Communication Technologies in Tourism 2016 2016 (pp. 621-635). Springer International Publishing. DOI: 10.1007/978-3-319-28231-2_45.

51. Ando Y, Thawonmas R, Rinaldo F. Inference of Viewed Exhibits in a Metaverse Museum. Culture and Computing (Culture Computing), 2013 International Conference on 2013 Sep 16 (pp. 218-219). IEEE. DOI: 10.1109/CultureComputing.2013.73.

52. Potkonjak V, Gardner M, Callaghan V, Mattila P, Guetl C, Petrović VM, Jovanović K. Virtual laboratories for education in science, technology, and engineering: A review. Computers & Education. 2016 Apr 30;95:309-27. DOI:10.1016/j.compedu.2016.02.002.

53. Project TimeTravel. http://www.projecttimetravel.com. Accessed: 2016-11-01.

54. Fischbach M, Latoschik ME, Bruder G, Steinicke F. smARTbox: out-of-the-box technologies for interactive art and exhibition. Proceedings of the 2012 Virtual Reality International Conference 2012 Mar 28 (p. 19). ACM. DOI: 10.1145/2331714.2331737.

APPENDIX

CODE METADATA	
Link to documentation/manual	https://sites.google.com/site/icelabuki
Software code languages, tools, and services used	C#
Operating environments	Windows 7 or newer
Compilation requirements & dependencies	Microsoft Kinect device

Table III. Atomic Postures and Type Conversion.

	Atomic Posture		Converted	
	name	value: description	type	description
1	2 Feet	0: Closed	Relative Joints	Left Foot is away from Right Foot less than 32 cm.
		1: Away from each other	Relative Joints	Left Foot is away from Right Foot at least 32 cm.
2	2 Hand	0: Closed	Relative Joints	Left Hand is away from Right Hand less than 35 cm.
		1: Away from each other	Relative Joints	Left Hand is away from Right Hand at least 35 cm.
3	Body	0: Not Jump/Crouch	Change-from-Initial	Spine moves up less than 7 cm.
			Change-from-Initial	Spine moves down less than 15 cm.
		1: Jump	Change-from-Initial	Spine moves up at least 7 cm.
		2: Crouch	Change-from-Initial	Spine moves down at least 15 cm.
4	Lean	0: Not Lean	Relative Joints	Center Shoulder is in front of Center Hip less than 4 cm.
			Relative Joints	Center Shoulder is behind Center Hip less than 14 cm.
		1: Forward	Relative Joints	Center Shoulder is in front of Center Hip at least 4 cm.
		2: Backward	Relative Joints	Center Shoulder is behind Center Hip at least 14 cm.
5	Left Hand [X]	1: Extend to the Left	Relative Joints	Left Hand is to the left of Center Shoulder at least 37 cm.
		2: Extend to the Right	Relative Joints	Left Hand is to the right of Center Shoulder at least 14 cm.
6	Left Hand [Y]	1: Raise Up	Relative Joints	Left Hand is above Center Shoulder at least 18 cm.
		2: Down	Relative Joints	Left Hand is below Center Shoulder at least 37 cm.
7	Left Hand [Z]	1: In Front of the Body	Relative Joints	Left Hand is in front of Center Shoulder at least 35 cm.
		2: Behind the Body	Relative Joints	Left Hand is behind Center Shoulder at least 9 cm.
8	Right Hand [X]	1: Extend to the Right	Relative Joints	Right Hand is to the right of Center Shoulder at least 37 cm.
		2: Extend to the Left	Relative Joints	Right Hand is to the left of Center Shoulder at least 14 cm.
9	Right Hand [Y]	1: Raise Up	Relative Joints	Right Hand is above Center Shoulder at least 18 cm.
		2: Down	Relative Joints	Right Hand is below Center Shoulder at least 37 cm.
10	Right Hand [Z]	1: In Front of the Body	Relative Joints	Right Hand is in front of Center Shoulder at least 35 cm.
		2: Behind the Body	Relative Joints	Right Hand is behind Center Shoulder at least 9 cm.
11	Left Leg	0: Stand	Relative Joints	Left Foot is in front of Center Hip less than 18 cm.
			Change-from-Initial	Left Knee moves up at less than 4 cm.
		1: Knee Strike	Relative Joints	Left Knee is in front of Center Hip at least 18 cm.
			Change-from-Initial	Left Knee moves up at least 15 cm.
		2: Kick	Relative Joints	Left Foot is in front of Center Hip at least 32 cm.
			Change-from-Initial	Left Knee moves up at least 4 cm.
12	Right Leg	0: Stand	Relative Joints	Right Foot is in front of Center Hip less than 18 cm.
			Change-from-Initial	Right Knee moves up at less than 4 cm.
		1: Knee Strike	Relative Joints	Right Knee is in front of Center Hip at least 18 cm.
			Change-from-Initial	Right Knee moves up at least 15 cm.
		2: Kick	Relative Joints	Right Foot is in front of Center Hip at least 32 cm.
			Change-from-Initial	Right Knee moves up at least 4 cm.