FAAST-R: Defining a Core Mechanic for Designing Gestural Interfaces

Evan A. Suma
Institute for Creative
Technologies
12015 Waterfront Drive
Playa Vista, CA 90094 USA
suma@ict.usc.edu

David M. Krum
Institute for Creative
Technologies
12015 Waterfront Drive
Playa Vista, CA 90094 USA
krum@ict.usc.edu

Belinda Lange
Institute for Creative
Technologies
12015 Waterfront Drive
Playa Vista, CA 90094 USA
lange@ict.usc.edu

Mark Bolas
Institute for Creative
Technologies
12015 Waterfront Drive
Playa Vista, CA 90094 USA
bolas@ict.usc.edu

Albert Rizzo
Institute for Creative
Technologies
12015 Waterfront Drive
Playa Vista, CA 90094 USA
rizzo@ict.usc.edu

Abstract

We present a syntax for representing human gestures using rule sets that correspond to the basic spatial and temporal components of an action. These individual rules form primitives that, although conceptually simple on their own, can be combined both simultaneously and in sequence to form sophisticated gestural interactions. Along with a graphical user interface for custom gesture creation, this approach was incorporated into the Flexible Action and Articulated Skeleton Toolkit as a recognition module (FAAST-R). This toolkit can either be used to facilitate the development of motion-based user interfaces or to repurpose existing closed-source applications and games by mapping body motions to keyboard and mouse events. Thus, this work represents an important step towards making gestural interaction more accessible for practitioners, researchers, and hobbyists alike.

Author Keywords
Natural interaction; gesture; video games, middleware

ACM Classification Keywords
H.5.2 [Information interfaces and presentation]: User Interfaces; K.8.0 [Personal Computing]: General.

General Terms
Design, Human Factors

Introduction

Recent advances in low-cost depth sensing technology has led to a proliferation of consumer electronics devices that can sense the user's body motion. The release of the Microsoft Kinect in late 2010 has sparked the rapid formation of a large and active community that has explored a myriad of uses ranging from informal hobbyist "hacks" to scientific research projects and commercial applications. However, despite the widespread accessibility of full-body motion sensing devices, designing intuitive and powerful gestural interactions remains a challenge for developers. In general, though the Kinect holds the record for the fastest selling consumer electronics device in history, the sales of many commercial Kinect for Xbox 360 game titles have been poor, which has been partially attributed to the lack of well-designed games that integrate body motion seamlessly into the experience [6] [10]. Indeed, research has shown that performing physical arm movements and gestures can have a profound impact on the user's attitudinal and emotional responses to visual stimuli [5] [8] [13]. These observations point to the need for both a theory of "gesture design" as well as the tools to enable the creation and customization of gestural interactions for 3D user interfaces and interactive media.

An important motivator for our work is the application of video game technology towards advances in the areas of rehabilitation [7] and health [9]. While the clinical value of leveraging motion gaming technology has received increased recognition in recent years, these applications pose several notable challenges for designers. Unlike commercial games, body-based control in a clinical setting is not "one-size-fits-all," and must be customizable based on individual patient medical needs, range of motion, and motivation level. For example, a client with impaired arm movement would require a therapy game that encourages motion just outside the boundary of comfort, but not so far that achieving the required body pose becomes overly frustrating or impossible. Thus, the gestural interactions need to be designed on a per-client basis by the clinician, who often may not possess intimate technical knowledge and programming skills. Furthermore, these interactions need to be easily and immediately adjustable as the patient improves or encounters problems.

To address the challenge of facilitating intuitive and transparent gesture design, we present a syntax for representing complex gestural interactions using rule sets that correspond to the basic spatial and temporal components of an action. These "action primitives" are represented as plain English expressions so that their meaning is immediately discernible for both technical and non-technical users, and are analogous to interchangeable parts on an assembly line - generic descriptors that can be reused to replicate similar gestural interactions in two completely different end-user applications. To facilitate that goal, this representation was incorporated into the Flexible Action and Articulated Skeleton Toolkit [11] as part of a recognition module (FAAST-R). This toolkit provides a complete end-to-end framework that includes a graphical user interface for custom gesture creation, sensor configuration, skeletal tracking, action recognition, and a variety of output mechanisms to control end-user applications such as 3D user interfaces and video games. FAAST can either be used to support development of original motion-based user interfaces from scratch or to repurpose existing applications by mapping body motions to keyboard and mouse events, and thus represents an important step towards making gestural interaction design and development more accessible for practitioners, researchers, and hobbyists alike.

Previous Work

Analysis of human motion typically requires solving three non-trivial problems: detection, tracking, and behavior understanding [15]. In this paper, the software libraries from OpenNI and Microsoft Research provide both user detection and skeletal tracking, so FAAST subsequently focuses on recognizing the action being performed by the tracked user and generating an appropriate output. The quantity of literature focusing on action recognition is vast (see [14] for a review). While approaches from the computer vision and machine learning communities are often highly sophisticated, they may be too daunting for a non-technically oriented user such as a clinician to manipulate, and we suggest that they would often be treated as "black box" algorithms unless a sufficiently intuitive user interface could be provided to customize the action recognition. Thus, in this work, we employed a conceptually simple action representation to ensure that the interaction schemes would remain transparent and easily discernible for our target user population.

The initial version of FAAST was released in late 2010, shortly after the release of the Microsoft Kinect, and supported a very limited set of interactions, such as extending the arm directly forward to activate a key press [11]. After the initial interest in the toolkit, mouse control with the user's hand was subsequently added. The idea for generating virtual input events was inspired in part by GlovePIE, a programmable input emulator that maps signals from a variety of hardware devices such as the Nintendo Wiimote into keyboard, mouse, and joystick commands [2]. Another similar solution is Kinemote, which allows mouse control of Windows applications and games using a floating hand and a "Palm Click & Drag" metaphor [3]. However, to the best of our knowledge, FAAST is the first software toolkit that defines a simple yet powerful action syntax and provides an interface for non-technically oriented users to design gesture recognition schemes in an intuitive way.

System Overview

FAAST is designed as middleware between the depth-sensing camera skeleton tracking libraries and end-user applications. Currently supported hardware devices include the Microsoft Kinect sensor using the Microsoft Research Kinect SDK and OpenNI-compliant sensors using the NITE skeleton tracking library from PrimeSense. Figure 1 illustrates the toolkit's architecture. To make the toolkit as device agnostic as possible, communication with each skeleton tracker is split into separate modules that are dynamically selected and loaded at runtime. Each module reads data from its respective tracker into a generic skeleton data structure. Thus, FAAST is easily extensible by adding new modules to support future devices and software libraries.

Once a skeleton has been read from the sensor, FAAST then processes this data in an action recognition module. Based on the actions being performed, the core application then invokes an emulator module that sends simulated keyboard and mouse events to the active window. The core FAAST application includes a graphical interface that allows the user to create custom gesture schemes and bind these actions to specific emulator events. For example, the user may choose to send a "left arrow key down" event when the user leans to the left by a certain number of degrees. Thus, FAAST allows users to control off-the-shelf applications and games using body motion, even though these interfaces only accept input from standard keyboard and mouse setups. The methods for depicting actions and designing custom gestures are described in the following two sections.
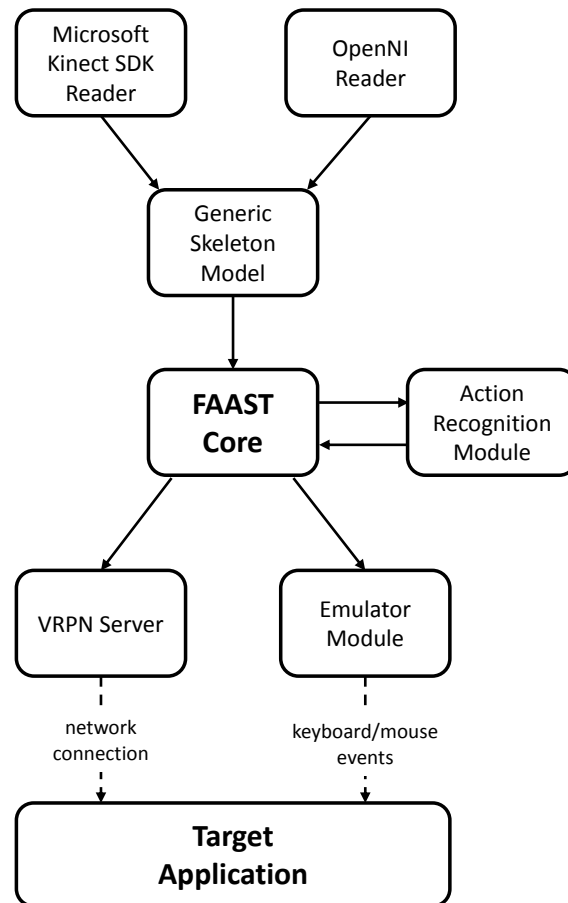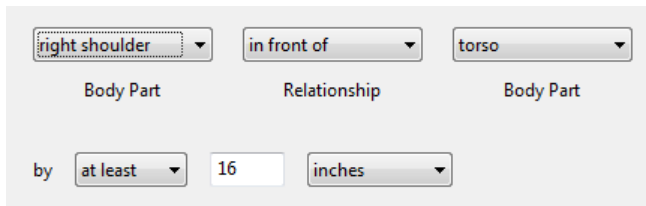
Figure 1: The FAAST architecture consists of modules that read skeleton data from depth sensing camera libraries, an action recognition module, an emulator module that sends simulated keyboard and mouse events to the active window, and a VRPN server to broadcast the position and orientation of each skeleton joint to networked applications.

In addition to simulating keyboard and mouse events, FAAST also streams the position and orientation of each skeleton joint using a Virtual Reality Peripheral Network (VRPN) server, which is a network interface that has become a popular standard for facilitating communication between VR applications and physical tracking devices [12]. This allows developers to read skeleton data into their applications in a device-independent manner using the same standard protocols that are commonly used by the community. FAAST is currently being officially supported by two commercial virtual world development environments for this purpose, 3DVIA Studio [1] and the Vizard virtual reality toolkit from WorldViz [4].

Decomposing Gestures

In order to enable users to design custom gestural interactions, we considered how to decompose complex body movements into atomic components. These simple action primitives form the conceptual "building blocks" that can be combined in FAAST to form more complicated gesture recognition schemes. To make these actions comprehensible and intuitive for non-technical users, they are formed by selecting terms from drop down boxes to form plain English expressions, similar to the way they might be described in everyday conversation (see Figure 2). For example, to design an action that activates when the user extends the left hand directly out in front of the body, the user would select parameters to form an expression such as: "left hand in front of torso by at least 16 inches." For each action specified, the user must also specify the *comparison* (either "at least" or "at most"), the numeric *threshold* for activation, and the *units* of measurement.

Figure 2: Rules are defined using drop-down boxes that form plain English expressions. These action primitives form the basis for more complicated gestures.

Position Constraints
Position constraints refer to the relative spatial relationship between any two skeletal joints, depicted as follows:

```
{body part} {relationship} {body part}
 by {comparison} [threshold] {units}
```

{body part} = head, neck, torso, waist, left shoulder, left elbow, left wrist, left hand, right shoulder, right elbow, right wrist, right hand, left hip, left knee, left ankle, left foot, right hip, right knee, right ankle, right foot

{relationship} = to the left of, to the right of, in front of, behind, above, below, apart from

{units} = centimeters, meters, inches, feet

Angular Constraints
It is also useful to consider cases when users flex or straighten one of their limbs, determined by calculating the angle of intersection between the two vectors connecting the limb's joint locations. Angular constraints are depicted as follows:

```
{limb} flexed
```

```
 by {comparison} [threshold] {units}
```

{limb} = left arm, right arm, left leg, right leg

{units} = degrees, radians

Velocity Constraints
Velocity constraints refer to the speed at which a particular body part is moving, depicted as follows:

```
{body part} {direction}
 by {comparison} [threshold] {units}
```

{body part} = head, neck, torso, waist, left shoulder, left elbow, left wrist, left hand, right shoulder, right elbow, right wrist, right hand, left hip, left knee, left ankle, left foot, right hip, right knee, right ankle, right foot

{direction} = to the left, to the right, forward, backward, up, down, in any direction

{units} = cm/sec, m/sec, in/sec, ft/sec

Body Constraints
In addition to the constraints listed above, it is also useful to consider body actions that are more "global," i.e. movements of the whole body relative to the camera, as opposed to positioning individual body parts relative to one another. We define two angular body actions, *lean* and *turn*, depicted as follows:

```
lean {left, right, forward, backward}
by {comparison} [threshold] {units}
```

```
turn {left, right}
by {comparison} [threshold] {units}
```

{units} = degrees, radians

Additionally, we also define a *jump* action. Measuring the height of a jump is not immediately obvious, however, because the height value of each skeleton joint is relative to the sensor, not the floor. Thus, to detect jumps, we leverage the fact that these actions occur very quickly, and consider the lowest height value of the feet over a previous window of time to be the height of the floor (experimentally determined to be 0.75 seconds). The jump action activates when both feet rise above this floor height value by the specified distance threshold. We found that when the user stands in one spot, which is frequently the case when interacting with depth sensors due to the restricted field of view, jump detection is quite reliable. Jump actions are depicted as follows:

```
jump by {comparison} [threshold] {units}
```

{units} = centimeters, meters, inches, feet

Time Constraints
The last type of constraint we consider is the temporal delay between the individual actions that constitute a gesture. In informal testing, we determined that it can be useful to be able to define action timing based on either previous action's start time or stop time, depending on the specific gesture being designed. Thus, time constraints are depicted as:

```
wait for [minimum] to [maximum] seconds
after action {starts, stops}
```

Designing Custom Gestural Interactions
The atomic actions described in the previous section are intentionally simple, and as a result the interaction possibilities provided by each individual constraint are limited. However, by combining these "action primitives" both simultaneously and in sequence, sophisticated

gestural interactions can be represented. Thus, we developed a graphical user interface for designing custom gestures using these atomic actions as conceptual building blocks (see Figure 3). New gestures are defined using a tree structure, with each gesture containing a list of input actions and a list of output events. Input actions and output events are added through dialog boxes with simple drop-down menus, and can be reorganized in the tree structure by clicking and dragging.
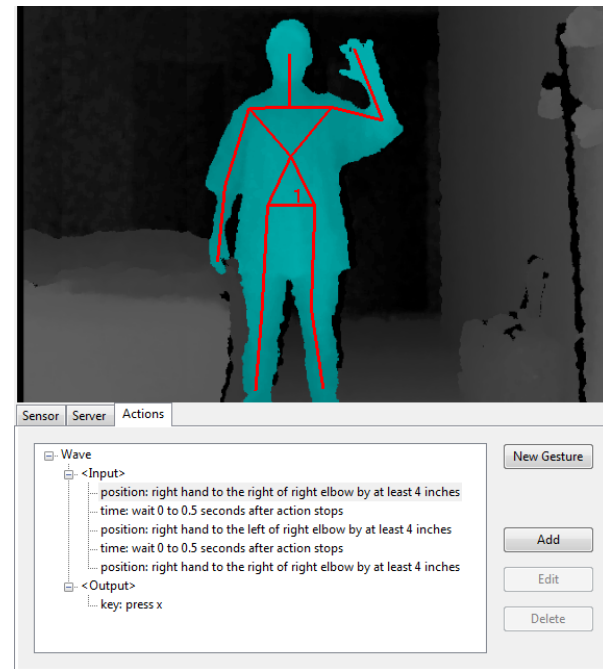

Figure 3: A screenshot of FAAST's gesture creation interface. The specified "wave" gesture activates when the user's right hand move rapidly back and forth relative to the elbow. When the action is detected, an 'x' key press event is sent to the selected end-user application.

When the emulator is running, FAAST queries the action recognition module to determine if all the input conditions are met, after which the emulator module is called to invoke the output events. To ensure that gesture does not reactivate too quickly, the user can also specify an optional timeout value. After a gesture activates, it will remain inactive until the timeout has elapsed.

Simultaneous Actions
Any input actions that are added without a time constraint separating them are treated as simultaneous events. The overall gesture will only be activated when all of the input conditions are simultaneously true at a given moment in time. For example, to create an interaction that requires the user to move both hands together in a quick "push" action out in front of the body, one could specify the following gesture:

```
position: right hand in front of torso
          by at least 16 inches
position: left hand in front of torso
          by at least 16 inches
position: right hand apart from left hand
          by at most 10 inches
velocity: right hand forward
          by at least 5 ft/sec
velocity: left hand forward
          by at least 5 ft/sec
```

Action Sequences
By combining position, angular, velocity, and body constraints simultaneously, users can finely tune body pose and movement for a given moment in time. However, many gestures require multiple actions to be performed in sequence. In FAAST, this behavior is represented using time constraints as separators between actions. When a time constraint is encountered, the action recognizer waits until the minimum time has elapsed (which may be zero), and then starts checking for the next group of simultaneous input actions to activate. If the actions do not activate before the maximum time has elapsed, the gesture resets and action recognition resumes from the very beginning of the sequence. Figure 3 shows an example of a "wave" gesture that uses sequential actions to detect when the user's right hand moves rapidly back and forth relative to the elbow.

Discussion
The combination of simultaneous and sequential actions allows users to design complex gestures that FAAST can recognize in real-time. There is no predetermined bound on the complexity or number of gestures that users may build, and so ultimately they are limited only by their own creativity. Furthermore, because the gestures are created by combining rules that are individually simple, the process of tweaking the parameters of the action is easily discernible. This is a notable advantage for non-technical users, such as physical therapy clinicians, who may not have the technical expertise to manipulate action recognition approaches that use computer vision or machine learning algorithms. However, this action representation may not be appropriate for more "organic" gestures that are not easily broken down into definable rule sets. Additionally, FAAST relies on two skeletal tracking libraries that do not have sufficient fidelity to segment fingers, so more subtle hand gestures such as sign language are not currently recognizable. As the fidelity of consumer depth-sensing technology matures, we expect to add new sensor modules that will support a wider range of hardware devices, so this limitation may be surmountable in the future.

Conclusion and Future Work

In this paper, we describe a core mechanic for designing and customizing gestural interactions and present an integrated toolkit that enables existing games and applications to be repurposed for body-based control. In the near future, we will begin evaluating the gesture creation interface with clinicians to gather feedback on its usability for rehabilitation. Additionally, there are many ways the toolkit may be extended and improved. For example, instead of entering the threshold values manually, the user will be able to pose in front of the sensor, and these values will be calculated automatically. We also plan to implement other types of action representations that may augment our base rule set, such as depicting keyframes with a poseable model.

FAAST is free software. The most exciting consequences that we observed when the toolkit was initially released were the novel uses of the toolkit by members of the community. Building on FAAST's existing user base, we believe that our easy-to-understand rule-based gesture creation paradigm will further empower hobbyists, researchers, and practitioners to design rich gestural interactions in novel ways that we have not expected.

References

[1] http://www.3dvia.com/studio/.

[2] http://www.glovepie.org/.

[3] http://www.kinemote.net/.

[4] http://www.worldviz.com/products/vizard/.

[5] J. T. Cacioppo, J. R. Priester, and G. G. Berntson. Rudimentary Determinants of Attitudes. II: Arm Flexion and Extension Have Differential Effects on Attitudes. *Journal of Personality and Social Psychology*, 65(1):5–17, July 1993.

[6] D. Hughes. Microsoft Kinect shifts 10 million units, game sales remain poor. *Huliq*, Mar. 2011.

[7] B. Lange, E. A. Suma, B. Newman, T. Phan, C.-y. Chang, A. Rizzo, and M. Bolas. Leveraging Unencumbered Full Body Control of Animated Virtual Characters for Game-Based Rehabilitation. In *HCI International*, pages 243–252, 2011.

[8] B. P. Meier and M. D. Robinson. Why the Associations Between Affect and Vertical Position. *Psychological Science*, 15(4):243–247, 2004.

[9] A. Rizzo, B. Lange, E. A. Suma, and M. Bolas. Virtual reality and interactive digital game technology: new tools to address obesity and diabetes. *Journal of Diabetes Science and Technology*, 5(2):256–264, Jan. 2011.

[10] S. Stein. Kinect, 2011: Where art thou, motion? *CNET*, June 2011.

[11] E. A. Suma, B. Lange, A. Rizzo, D. M. Krum, and M. Bolas. FAAST : The Flexible Action and Articulated Skeleton Toolkit. In *IEEE Virtual Reality*, pages 245–246, 2011.

[12] R. M. Taylor, T. C. Hudson, A. Seeger, H. Weber, J. Juliano, and A. T. Helser. VRPN: a device-independent, network-transparent VR peripheral system. In *ACM Virtual Reality Software and Technology*, pages 55–61, 2001.

[13] D. Tucker. *Towards a theory of gesture design*. M.F.A. thesis, University of Southern California, 2012.

[14] P. Turaga, R. Chellappa, V. S. Subrahmanian, and O. Udrea. Machine Recognition of Human Activities: A Survey. *IEEE Transactions on Circuits and Systems For Video Technology*, 18(11):1473–1488, 2008.

[15] L. Wang, W. Hu, and T. Tan. Recent developments in human motion analysis. *Pattern Recognition*, 36(3):585–601, Mar. 2003.