Development of a Hand Gestures SDK for NUI-Based Applications

Seongjo Lee, Sohyun Sim, Kyhyun Um, Young-Sik Jeong, Seung-won Jung, and Ky ungeun Cho

Department of Multimedia Engineering, Dongguk University-Seoul, Seoul 100-715, Republic of Korea

Correspondence should be addressed to Kyungeun Cho; cke@dongguk.edu

Received 18 August 2014; Accepted 18 October 2014

1. Introduction

The major advantage of ubiquitous computing is that it enables users to use computers and networks in natural and intuitive ways. Furthermore, ubiquitous computing is characterized by the use of human-focused interfaces, as opposed to computer-focused interfaces using existing input devices. Accordingly, research related to ubiquitous concepts has been playing a significant role in realizing the future of computing, in which computers unobtrusively support humans in everyday life.

In this context, human computer interaction (HCI) is being actively investigated to facilitate the implementation of human-focused computer environments. In particular, research into approaches that enable computers to recognize and understand users’ inputs and provide corresponding services has been gaining attention. Heretofore, general interaction between humans and computers has primarily been achieved using intermediary devices such as keyboard and mouse. However, one of the major disadvantages of interaction using such input devices is the fact that users have to learn how to use them. In addition, spatial limitations exist because the devices have to be directly connected to a computer. Consequently, natural user interface (NUI) research is being actively conducted to supplement existing interaction approaches [1, 2].

NUI is human-focused interface that uses the human body as an input, using sensors or cameras [3]. In the NUI approach, users interact with computers via natural human traits such as gestures, viewpoints, and face tracking and recognition [4, 5].

The interaction approach using gestures facilitates a more natural interface and enables users to intuitively use computer systems [6]. Furthermore, it also facilitates intuitive use of the system such that users do not have to learn how to use the interface and input devices [7]. Subsequent to the development of the Kinect sensor, which recognizes depth information and human joint information, by Microsoft’s “Project Natal” at the end of 2010, research into the recognition of gestures using depth information and human joint information has been actively pursued [8, 9]. Further, research is also being conducted into how human centric interfaces can be used to recognize situations and intentions for a flexible interface between humans and machines, such as conversations between human beings [10].
The hand gesture SDK proposed in this paper identifies hand gestures using vector chain code. The screen area is divided into a grid to identify the vector chain code and gestures are recognized using the identified vector chain code and HMM. The proposed approach shows the identification area on the screen and thereby enables users to obtain feedback on the execution of a gesture by watching the screen. Furthermore, this system is suitable for application development because gestures are quickly recognized.

NUI-based application development currently requires special NUI-related knowledge. Consequently, developers who have no basic knowledge of NUI have difficulty developing NUI-based applications. This paper proposes a hand gesture SDK that facilitates development of NUI-based applications without any specific NUI knowledge. We present the results obtained using the proposed SDK to develop a “Smart Interior” system comprising a virtual environment and contents.

The remainder of this paper is organized as follows: Section 2 introduces existing research related to the recognition of hand and arm gestures. Section 3 outlines the overall structure of the gesture SDK proposed in this paper. Section 4 presents and discusses our implementation results for the “Smart Interior” application using the proposed gesture SDK. Finally, Section 5 concludes this paper.

2. Related Work

This paper proposes a hand gesture SDK that uses joint information from the Kinect sensor. Approaches for recognizing hand gestures have been extensively investigated over the years. Further, the use of hand gestures as input is being studied for a variety of applications and games in various fields, including the medical field.

For example, Kim et al. [1] proposed an algorithm that controls multimedia content solely by recognizing human gestures. The method detects the hands on the basis of the depth image of Kinect and YCbCr color image. The hand trace is then translated into eight-direction chain code and the hand gestures are consequently recognized by hidden Markov model (HMM) algorithm.

Park et al. [3] proposed a hand gesture recognition approach that uses visual images and depth information. In the proposed approach, the hand area is detected in the depth image obtained from a Kinect sensor. Then, the hand area in the depth information and the visual image are mapped. Consequently, hand gestures are recognized by defining the number of fingers unfolded. They assessed the performance of their proposed system by investigating its ability to control a medical image on a monitor.

Heo et al. [6] presented an approach based on HMM algorithm that recognizes one-arm gestures in real time environments. In their approach, they configure a vector using the coordinates of arm joints detected by Kinect. Next, they execute a feature transformation process to convert the vector into the angle value between arm joints. After dimension reduction and discretization through k-means clustering, the arm gestures are then recognized using HMM. Lee and Choi [11] applied mathematical morphology to gesture recognition. Their approach traces the trajectory of the hand centers in hand gesture sequences containing important data related to the form of the hand gesture. The hand gesture sequences are then recognized by acquiring eight-direction chain code edge vectors from the traced trajectory of centers.

The NUI approaches presented above use human gestures as input data in applications, games, and the medical field. However, they did not develop an SDK, which could aid in the development of programs, for recognition of hand or arm gestures. This paper proposes a gesture SDK that enables developers to develop NUI-based applications more easily.

3. The Gesture SDK

The gesture SDK proposed in this paper comprises the Gesture Recognition Component, which recognizes gestures in applications, and the Custom-Defined Gesture Creator, which generates the gestures used in applications.

3.1. The Custom-Defined Gesture Creator. The gesture SDK proposed in this paper classifies gestures as either one-handed or two-handed gestures. It provides five kinds of basic one-handed gestures and two kinds of two-handed gestures as predefined gestures. Figures 1(a) and 1(b) show the various predefined one-handed and two-handed gestures, respectively.

The Kinect Interaction Module in this system classifies the beginning and end of a gesture into grip and nongrip and gesture and nongesture. Consequently, this system can classify the gestures entered by a user and differentiate them from other gestures.

The Custom-Defined Gesture Creator enables developers to directly generate custom gestures for implementing a variety of NUI-based applications. It extracts feature vectors from human gestures and generates an HMM to recognize the gesture using the extracted feature vector.
For $i = 1$ to $k$

Start

$K =$ number of learning
State $=$ number of inflection points
Name $=$ gesture name
Id $=$ gesture ID

Gesture input

Learning relevant data

Save features, $i++$

No

Yes

HMM generation

**Figure 2: Custom-Defined Gesture Creator flowchart.**

It comprises a Kinect Interaction Module and a Gesture Maker Module. The Kinect Interaction Module loads the human hand data captured by Kinect and extracts the features of the gesture from the loaded data. The Gesture Maker Module then generates an HMM to recognize the gesture using the features extracted by the Kinect Interaction Module.

The program generates gestures on receiving the gesture ID number required for defining the gesture, the gesture name, the number of inflection points classified into eight directions, and the gesture learning number. Figure 2 presents the one-handed gesture HMM generation flowchart using the Custom-Defined Gesture Creator program.

### 3.1.1. The Kinect Interaction Module

To recognize gestures, the proposed system classifies human motions into gestures and nongestures. A gesture is a hand motion for controlling a program, whereas a nongesture is meaningless motion between gestures. This module classifies motions using the hand status data from the Kinect SDK and the Kinect sensor. In the system, a clenched fist is classified as a gesture, whereas an open hand is deemed a nongesture.

The Kinect Interaction Module converts the trajectory of gestures into eight-directional vector chain code and trains the HMM using the chain code as the features for recognizing the gestures. It acquires joint location data from Kinect in order to obtain the trajectory of the hand, which is used as the features of the gestures. After getting the data to extract the features of gestures from Kinect, the Kinect Interaction Module extracts a feature vector to train the HMM to recognize gestures. The Kinect Interaction Module is also utilized to carry out this function in the Gesture Recognition Component.

To effectively use the features of applications using screens, the proposed method adopts the screen for recognizing the gestures. The Kinect Interaction Module extracts the feature vector by converting hand coordinates into screen coordinates and dividing the screen space into several grids. Next, the hand coordinates on the screen are quantized on grids. The features of the gestures are then extracted using the quantized grids. The hand gesture recognition field can be moved to the screen using the proposed approach. Further, users can give feedback on the gestures while watching the hands on the screen. Figure 3 illustrates the data flow in the Kinect Interaction Module.

Eight-direction vector codes range from one to eight in a clockwise direction, with the top vector as the basis. Extraction of the eight-direction vector chain code comprises preparation for extracting the features, storage of the input data, and extraction of features. Figure 4 presents the three steps used to extract the features.
First, the screen is segmented into grid cells; in this case, the screen is segmented into $10 \times 6$ grid cells. The location data of the 3D hand joint is then projected onto 2D screen coordinates.

Next, the system quantizes the hand coordinates entered in sequence into the segmented grid cells. If the grid cell for the hand coordinates in the previous frame is the same as that in the present frame, the present grid cell is excluded to eliminate overlapping data in the quantized grid cell. The quantized grid cells are then saved in sequence following elimination of the overlapped data. Figure 5 shows an example on a $3 \times 3$ grid field that explains the second process outlined above. The dots in the first figure are 2D hand joint coordinates in accordance with the passage of time. The shaded grid cells in the second figure show the grid cells through which the hand joints have passed. Grid cells 5, 8, and 9 are saved in sequence.

Next, the relative vectors of the grid cells saved in sequence are calculated in relation to the previous location and converted to vector chain code. During the calculation of the relative vector using the rectangular grid field, the upward and downward movement and movement to the left and right can be accurately extracted because the surface area resulting from the grid cell meeting other grid cells above it, below it, on the left, and on the right is large. However, the grid cells located diagonally meet only at each grid’s vertex. Consequently, accurate extraction of diagonal movement is difficult. Figure 6 illustrates the movement of a hand in the grid field in the diagonal direction.

As shown in Figure 6, when a hand moves to the vector in the number 2 direction, the grid cells through which the hand joint coordinates pass are saved in the order of 5, 2, and 3 or 5, 6 and 3 not in the order of 5 and 3.

In this case, while the hand actually moved to the vector in the number 2 direction, the vector for the relative location in the grid field is extracted as the combination of two vectors, 1-3 or 3-1. To rectify this problem, when a hand moves in the diagonal direction, the vector corresponding to the relevant diagonal direction conducts the extraction by considering the grid cell next to the cell in question. Figure 7 presents a flowchart for extraction of the eight-direction vector using the relative location of the grid cell saved.

3.1.2. The Gesture Maker Module. HMM is ideal for dealing with sequential data and is consequently frequently used to recognize gestures and voice in NUI fields. HMM has a variety of models depending on the direction of state transition. These include the Ergodic model and the Left-Right model. This paper adopts the Left-Right HMM to
recognize the hand gestures using the feature vector extracted by the Kinect Interaction Module.

The definition of the number of Left-Right HMM states varies according to the gesture patterns. The number of predefined gesture states ranges between two and eight and is equal to the number of inflection points. The number of HMM states is represented by \( N \), and \( S \) represents a set of states:

\[
S = \{s_1, s_2, \ldots, s_N\}. \tag{1}
\]

HMM has three parameters:

\[
\theta = (A, B, \pi), \tag{2}
\]

in which \( A \) is a state transition probability matrix:

\[
A = [a_{ij}], \quad (1 \leq i, j \leq N), \quad a_{ij} = P(q_{t+1} = s_j \mid q_t = s_i), \tag{3}
\]

\[
\sum_{j=1}^{N} a_{ij} = 1,
\]

where \( q_t \) is the state of the model at \( t \).

\( B \) is an observation symbol probability matrix:

\[
B = [b_j(v_k)], \quad v_k \in V
\]

\[
O = \{o_1, o_2, \ldots, o_L\}, \quad (1 \leq t \leq L)
\]

\[
b_j(v_k) = P(o_t = v_k \mid q_t = s_j)
\]

\[
\sum_{k=1}^{8} b_j(v_k) = 1,
\]

where \( V \) is a set of 8-direction vectors, \( O \) is a vector chain code, \( L \) is the length of \( O \), and \( o_t \) is an observation vector at \( t \).

\( \pi \) is a vector of initial state probabilities:

\[
\pi = [\pi_i], \quad (1 \leq i \leq N), \quad \pi_i = P(q_1 = s_i). \tag{5}
\]

To generate the HMM, the three HMM parameters are optimized using extracted 8-directional vector chain codes. The learning algorithm used in this paper is the Baum-Welch algorithm. The Baum-Welch algorithm, an expectation and maximization (EM) algorithm, is typically used to train HMMs. The algorithm uses two latent variables; it comprises estimating latent variables (E step) and updating parameters, \( A \), \( B \), and \( \pi \) (M step) [12].

However, when this algorithm is utilized, the possibility exists that the three parameters may become local optimums. To prevent this from happening, we generate the initial values of the parameters using a random number generator and repeat the process 150 times. When the random number generator assigns \( A = |a_{ij}| \), the condition of \( A \) is given by

\[
a_{ij} = 0, \quad (i > j). \tag{6}
\]

As stated above, we adopt Left-Right HMM. Figure 8 presents the state transition diagram for Left-Right HMM.

Fifty learning datasets were used per gesture to train the predefined gesture HMM; recognition accuracy increases with the amount of learning data used.

The models learned by the Gesture Maker Module are saved in “Gesture DB”—a repository containing the gestures that the proposed SDK can recognize. The models saved in “Gesture DB” are used to recognize the user gestures in the Gesture Recognition Component.

The proposed system supports both two-handed and one-handed gestures. The process used to generate two-handed gestures does not require the generation of a new model. Because a two-handed gesture is recognized as a combination of two one-handed gestures, a new ID is defined for a two-handed gesture by inputting the one-handed gesture ID for the left hand and that for the right hand.

3.2 The Hand Gesture Recognition Component. The Gesture Recognition Component acts as an intermediary between the application layer and users. It comprises the Kinect Interaction Module and the Gesture Recognition Manager Module.

The Kinect Interaction Module is the same as that described for the Custom-Defined Gesture Creator above. The Gesture Recognition Manager Module recognizes users’ gestures and sets the gestures the application can recognize. Figure 9 depicts the architecture of the Gesture Recognition Component.
The Gesture Setup Module in the Gesture Recognition Manager sets gestures that can be recognized in the application. Developers can use “Gesture Setup” to change the kinds of gestures depending on the application state. The “Gesture Recognizer DB” registers the models of gestures for recognition in the application. When a developer enters the list of gestures to recognize some gestures using “Gesture Setup,” the HMM of the gestures registered in “Gesture DB” in the Custom-Defined Gesture Creator is registered in “Gesture Recognizer DB.”

With the “Gesture Recognizer DB,” only the HMM of the gesture to be recognized is compared. This approach reduces the frequency of false recognition by the Gesture Recognizer as compared to the comparison containing the HMM, even for gestures not used in the application. In consideration of the features of the “Gesture Recognizer DB,” one-handed and two-handed gestures are saved in different places in the repository. Furthermore, even the repository for the two-handed gesture is divided into two spaces—one for the left hand and the other for the right hand.

Gesture Recognizer classifies the gestures when a user’s gesture is entered and generates the corresponding events. In the HMMs of the gesture registered in the “Gesture Recognizer DB,” the likelihoods are computed using the vector chain extracted by the Kinect Interaction Module.

Each model evaluates the probability with which a vector chain code identified from a user’s gesture is entered. The gesture model with the highest such probability is classified as the gesture of the identified vector chain code. This is given by the expression

\[ q = \arg \max_i P(O | \theta_{g_i}), \]

where \(O\) is the vector chain of a hand motion, \(\theta_{g_i}\) is the HMM of the gesture learned, \(q\) is an estimation gesture index, and \(P(O | \theta)\) is calculated using the Forward algorithm [12].

When the left and right hand IDs entered when defining two-handed gestures in the Custom-Defined Gesture Creator match, the relevant two-handed gesture is recognized.

When left and right hand IDs do not match, input gesture is rejected.

Subsequently, the event function containing the result of gesture recognition is called. Developers develop applications using the results of gesture recognition by registering the callback function to the event.

4. Experiment and Analysis

This section introduces “Smart Interior,” a NUI-based application developed using the gesture SDK proposed in this paper. Further, the performance of the gesture SDK in the “Smart Interior” application is evaluated. This SDK is developed using C# language.

4.1. Smart Interior. Smart Interior is used to decorate a virtual house by arranging furniture or accessories or changing floor materials, ceiling, or wallpaper. Further, users can enter and look around the house in a first person point of view. Additional gestures were generated using the Custom-Defined Gesture Creator, the SDK tool, for the gestures required in “Smart Interior.” Figure 10 shows the gestures used in “Smart Interior,” while Figure 11 illustrates the steps used to add the gestures using the Custom-Defined Gesture Creator.

The application classifies the purpose of gestures using the status of the left and right hands and whether a hand is open or closed, when recognizing one-handed gestures. The gesture made with the left hand closed is the System Gesture, which controls system components such as Confirm, Cancel, Undo, Redo, Save, and View Mode. The gesture made with the right hand closed is the Menu Control Gesture to control the list of items on the interior menu. Two-handed gestures are used to control the camera view and edit the interior items. It executes scale and rotate for items and sets the camera view to TopView or QuarterView.

The “Smart Interior” application has four kinds of scenes for using gestures: “Start Scene,” “Select House Model,” “Load,” and “Edit Scene.” “Start Scene” is used to build a new house or load a previously saved house. “Select House Model” is used to select the house structure for new houses. “Load” is used to select and load previously decorated houses. “Edit Scene” is used to decorate houses. Because each scene uses different gestures, the gestures to be used in the relevant scene were defined using Gesture Setup in the Gesture Recognition Manager. Tables 1, 2, and 3 list the commands used to control Smart Interior.

The results above show that NUI-based applications can be developed using the gesture SDK proposed in this paper.
Table 2: Gestures used in “Select House Model” and “Load” and actions for gestures.

<table>
<thead>
<tr>
<th>Grip hand</th>
<th>Gesture</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>OK</td>
<td>Load the selected house</td>
</tr>
<tr>
<td>Right</td>
<td>Left</td>
<td>Go to the house on the left</td>
</tr>
<tr>
<td>Right</td>
<td>Right</td>
<td>Go to the house on the right</td>
</tr>
</tbody>
</table>

Figures 12 and 13 show the interaction with the application using the gestures defined in this application.

4.2. Performance Assessment. We used 50 images per gesture in our assessment of hand gestures recognition performance. The Kinect sensor used to obtain the images was installed at a height of 1 m. The distance between the Kinect sensor and a human being was 1.5 m–2 m when the pictures were being taken.

Table 4 presents the performance calculated using the approach proposed in this paper. $N$ is the number of input gestures, $N_{\text{rej}}$ is the number of rejected gestures, $N_{\text{Err}}$ is the number of gestures with recognition errors, $N_{\text{Hit}}$ is the number of gestures recognized properly, and $R$ is the recognition ratio:

$$R = \frac{N_{\text{Hit}}}{N} \times 100\%.$$  \hspace{1cm} (8)

To verify the real-time interaction with a computer using the recognition of gestures proposed in this paper, the recognition time of each gesture used in “Smart Interior” was measured 20 times. Table 5 presents the average performance time.

The default value for monitor refresh rate is predominantly 60 Hz. Thus, when the program executes the vertical synchronization in accordance with the default value above, the output time for one frame is 16.6667 ms. This result confirms that the program is applicable to real-time HCI.

5. Conclusion

This paper proposed “Gesture SDK” to facilitate the development of gesture recognition-based applications without special knowledge. Although various gesture recognition-related proposals exist, they are primarily limited to identification of and recognition of gestures, in contrast to our proposed SDK for developing NUI-based applications. Using the proposed SDK, developers can directly define gestures.
and simply apply the defined gestures to their applications. Our gesture recognition experiment for the “Smart Interior” developed using the SDK indicates an average recognition ratio of 94%. The results suggest that NUI-based applications can be developed using the proposed “Gesture SDK.”

The gesture generator, Custom-Defined Gesture Creator, proposed in this paper currently cannot identify duplicated gestures. Further study is therefore needed in order to develop a method for recognizing duplicated gestures. Furthermore, the algorithm that automatically sets the state value when defining the gestures also requires more research. A method of differentiating gestures from user’s motion without using grip and release is also needed.

**Conflict of Interests**

The authors declare that there is no conflict of interests regarding the publication of this paper.
Table 3: Gestures used in “Edit Scene” and actions for gestures.

<table>
<thead>
<tr>
<th>Grip hand</th>
<th>Gesture</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Right</td>
<td>Menu</td>
<td>Open the list of furniture/items</td>
</tr>
<tr>
<td>Right</td>
<td>Right</td>
<td>Open the list of furniture/items on the right</td>
</tr>
<tr>
<td>Right</td>
<td>Left</td>
<td>Open the list of furniture/items on the left</td>
</tr>
<tr>
<td>Right</td>
<td>Up</td>
<td>Go to the next page of the list of furniture/items</td>
</tr>
<tr>
<td>Right</td>
<td>Down</td>
<td>Go to the previous page of the list of furniture/items</td>
</tr>
<tr>
<td>Left</td>
<td>Left</td>
<td>Undo</td>
</tr>
<tr>
<td>Left</td>
<td>Right</td>
<td>Redo</td>
</tr>
<tr>
<td>Left</td>
<td>View</td>
<td>Look around the house in the process of decoration</td>
</tr>
<tr>
<td>Left</td>
<td>Save</td>
<td>Save the house in the process of decoration</td>
</tr>
<tr>
<td>Left/right</td>
<td>Scale up</td>
<td>Increase the size of the selected furniture/item</td>
</tr>
<tr>
<td>Left/right</td>
<td>Scale down</td>
<td>Reduce the size of the selected furniture/item</td>
</tr>
<tr>
<td>Left/right</td>
<td>Rotate left</td>
<td>Rotate the selected furniture/item counterclockwise</td>
</tr>
<tr>
<td>Left/right</td>
<td>Rotate right</td>
<td>Rotate the selected furniture/item clockwise</td>
</tr>
<tr>
<td>Left/right</td>
<td>Top view</td>
<td>Change the camera view to top</td>
</tr>
<tr>
<td>Left/right</td>
<td>Quarter view</td>
<td>Change the camera view to quarter</td>
</tr>
</tbody>
</table>

Table 4: Gesture performance assessment.

<table>
<thead>
<tr>
<th>Gesture</th>
<th>$N_e$</th>
<th>$N_{rej}$</th>
<th>$N_{hit}$</th>
<th>$R_{hit}$</th>
<th>$R_{rej}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Right</td>
<td>50</td>
<td>0</td>
<td>50</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Left</td>
<td>50</td>
<td>0</td>
<td>45</td>
<td>90%</td>
<td>90%</td>
</tr>
<tr>
<td>Up</td>
<td>50</td>
<td>0</td>
<td>50</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Down</td>
<td>50</td>
<td>0</td>
<td>50</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>OK</td>
<td>50</td>
<td>0</td>
<td>50</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Menu</td>
<td>50</td>
<td>0</td>
<td>49</td>
<td>98%</td>
<td>98%</td>
</tr>
<tr>
<td>Save</td>
<td>50</td>
<td>0</td>
<td>47</td>
<td>94%</td>
<td>94%</td>
</tr>
<tr>
<td>View</td>
<td>50</td>
<td>0</td>
<td>50</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Scale up</td>
<td>50</td>
<td>1</td>
<td>49</td>
<td>98%</td>
<td>98%</td>
</tr>
<tr>
<td>Scale down</td>
<td>50</td>
<td>1</td>
<td>49</td>
<td>98%</td>
<td>98%</td>
</tr>
<tr>
<td>Top view</td>
<td>50</td>
<td>2</td>
<td>48</td>
<td>96%</td>
<td>100%</td>
</tr>
<tr>
<td>Quarter view</td>
<td>50</td>
<td>2</td>
<td>47</td>
<td>94%</td>
<td>95.92%</td>
</tr>
<tr>
<td>Rotate right</td>
<td>50</td>
<td>3</td>
<td>47</td>
<td>94%</td>
<td>100%</td>
</tr>
<tr>
<td>Rotate left</td>
<td>50</td>
<td>4</td>
<td>46</td>
<td>92%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Acknowledgments

This research was supported by the Ministry of Science, ICT, and Future Planning (MSIP), Korea, under the Information Technology Research Center (ITRC) support program (NIPA-2014-H0301-I4-1021) supervised by the National IT Industry Promotion Agency (NIPA).

Table 5: Average gesture recognition performance time.

<table>
<thead>
<tr>
<th>Gesture</th>
<th>$T_{ext}$ (ms)</th>
<th>$T_{recog}$ (ms)</th>
<th>$T_{total}$ (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Right</td>
<td>6.0026</td>
<td>1.6967</td>
<td>7.6993</td>
</tr>
<tr>
<td>Left</td>
<td>7.4087</td>
<td>1.0049</td>
<td>8.4136</td>
</tr>
<tr>
<td>Up</td>
<td>7.8124</td>
<td>2.0002</td>
<td>9.8126</td>
</tr>
<tr>
<td>Down</td>
<td>7.0095</td>
<td>2.3000</td>
<td>9.3095</td>
</tr>
<tr>
<td>OK</td>
<td>5.3028</td>
<td>1.4005</td>
<td>6.7033</td>
</tr>
<tr>
<td>Menu</td>
<td>7.9049</td>
<td>1.7017</td>
<td>9.6066</td>
</tr>
<tr>
<td>Scale up</td>
<td>10.9095</td>
<td>1.1981</td>
<td>12.1076</td>
</tr>
<tr>
<td>Scale down</td>
<td>10.9058</td>
<td>1.4999</td>
<td>12.4057</td>
</tr>
<tr>
<td>Rotate left</td>
<td>11.9065</td>
<td>1.7012</td>
<td>13.6077</td>
</tr>
<tr>
<td>Rotate right</td>
<td>11.8101</td>
<td>1.6002</td>
<td>13.4103</td>
</tr>
<tr>
<td>Top view</td>
<td>13.2083</td>
<td>2.3996</td>
<td>15.6079</td>
</tr>
<tr>
<td>Quarter view</td>
<td>13.6069</td>
<td>1.7022</td>
<td>15.3091</td>
</tr>
<tr>
<td>Total</td>
<td>9.5657</td>
<td>1.6837</td>
<td>11.2494</td>
</tr>
</tbody>
</table>

References

