Research Article

Method for User Interface of Large Displays Using Arm Pointing and Finger Counting Gesture Recognition

Hansol Kim, Yoonkyung Kim, and Eui Chul Lee

Department of Computer Science, Sangmyung University, Seoul 110-743, Republic of Korea

Correspondence should be addressed to Eui Chul Lee; eclee@smu.ac.kr

Received 27 June 2014; Accepted 15 August 2014; Published 1 September 2014

Academic Editor: Young-Sik Jeong

Copyright © 2014 Hansol Kim et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Although many three-dimensional pointing gesture recognition methods have been proposed, the problem of self-occlusion has not been considered. Furthermore, because almost all pointing gesture recognition methods use a wide-angle camera, additional sensors or cameras are required to concurrently perform finger gesture recognition. In this paper, we propose a method for performing both pointing gesture and finger gesture recognition for large display environments, using a single Kinect device and a skeleton tracking model. By considering self-occlusion, a compensation technique can be performed on the user's detected shoulder position when a hand occludes the shoulder. In addition, we propose a technique to facilitate finger counting gesture recognition, based on the depth image of the hand position. In this technique, the depth image is extracted from the end of the pointing vector. By using exception handling for self-occlusions, experimental results indicate that the pointing accuracy of a specific reference position was significantly improved. The average root mean square error was approximately 13 pixels for a 1920×1080 pixels screen resolution. Moreover, the finger counting gesture recognition accuracy was 98.3%.

1. Introduction

A significant amount of research has been conducted on hand gesture recognition. To perform interactive navigation and manipulation, pointing gesture and finger gesture recognition should be simultaneously executed.

Pointing gesture recognition methods can be categorized into two method types: two-dimensional (2D) image-based methods and three-dimensional (3D) methods. Although 2D image-based methods, dating back several decades, can be easily implemented today, their targeting accuracies are poor in comparison to more recent 3D methods. Therefore, 2D image-based methods are not considered in this paper.

Since the development of low cost, high depth perception 3D cameras, such as the Bumblebee and Kinect, 3D-based pointing gesture recognition methods have been widely researched. Yamamoto et al. proposed a real-time arm pointing gesture recognition method using multiple stereo cameras [1]. Because multiple stereo cameras cover a relatively wide area, the degree of freedom of a user's movement is relatively high. However, the calibration required to

define epipolar geometric relations among multiple stereo cameras is considerably expensive. Other methods [2, 3] have considered head orientation to accurately estimate the hand pointing position. Head orientation typically changes as the hand targeting position changes. However, head orientation data cannot be reliably obtained, which can degrade the accuracy of the estimated hand targeting position. Another method [4] approached this problem by analyzing appearance, interactive context, and environment. However, the individual variations of these additional parameters can also lead to decreased targeting accuracy.

Recently, pointing gesture methods based on the skeleton model of the Kinect SDK (Software Development Kit) have been reported [5]. One particular pointing gesture method that was proposed utilized the skeleton model and a virtual screen [6]. The critical issue in this method, however, was defining a correspondence between the virtual screen and the physical display. In addition, this method did not consider self-occlusion; it did not specifically address the issue of distinguishing both hand and shoulder points on a perspective line. Other 3D-based methods [7, 8] have also

FIGURE 1: Flow diagram of the proposed method.

failed to address this issue. Although 3D-based methods are accurate in terms of defining a pointing vector for a fingertip, unstable dithering problems caused by low-resolution images can occur when a camera is positioned at a distance [9].

To facilitate interactive display manipulation, many finger gesture recognition methods have been studied. In a previous research effort [11], a fingertip detection method that combined depth images with color images was proposed. In this method, a finger outline tracking scheme was used, and its accuracy was relatively high. However, because the operational distance between the camera and hand was relatively short, the method cannot be considered in our large display and long distance environment. An appearance-based hand gesture recognition method using PCA (Principal Component Analysis) was described [12]. However, this method presents problems such as illuminative variation and hand orientation, which are similar to problems observed in PCA-based face recognition. In an alternative approach, a 3D template-based hand pose recognition method was proposed [13]. In this method, a 2D hand pose image was recognized by comparing 26 DOF (Degree of Freedom) 3D hand pose templates. However, the method is tightly coupled with a predefined 3D hand pose template. In addition, the computational complexity for estimating 3D hand poses from the captured 2D image stream was high. In a current research, a new hand posture recognition method was proposed based on the sparse representation of multiple features such as gray-level, texture, and shape [14]. However, this method is strongly dependent on a training database. Furthermore, the binary decision for each feature's sparsity presents a problem, because continuous values of sparse features must be considered.

To solve the problems related to previous pointing and hand gesture methods, a new arm pointing and finger counting gesture recognition method is proposed in this paper. Our proposed method is a user-dependent, calibration-free method based on the Kinect skeleton model. We resolve the self-occlusion problem in the arm pointing gesture recognition module. Moreover, finger counting gesture recognition is accurately performed using a low-resolution depth image. Both gesture recognition techniques are performed with a single Kinect device.

2. Proposed Method

Our proposed method is executed as per the steps shown in Figure 1. The method is organized into two parts, namely, arm pointing gesture recognition and finger counting gesture recognition.

2.1. Arm Pointing Gesture Recognition. Arm pointing gesture recognition is performed using the sequence shown in the red dotted box of Figure 1. First, 3D coordinates of the right-hand and shoulder positions are obtained using the skeleton model of the Kinect SDK. In the visible image captured from the Kinect device, the X and Y values of an arbitrary pixel's 3D coordinates are the same as their corresponding pixel coordinates in the visible image. The Z value, which is measured by the Kinect's depth camera, is multiplied by 10 mm. Next, we proceed to step (b), in which the Euclidean distance between the shoulder position in the previous frame and the hand position in the current frame is measured. When both the hand and shoulder positions lie on the same

FIGURE 2: Examples of shoulder point detection errors caused by the self-occlusion problem (red dot: shoulder point, yellow dot: hand point).

camera perspective line, the shoulder position cannot be accurately detected because of occlusion by the hand, as shown in Figure 2. We use exception handling to address such self-occlusion; if the distance measured in step (b) of Figure 1, specifically, is less than the empirically defined threshold ($T = 10$ pixels), the current shoulder position is set to that of the previous frame (step (c)). If the distance is greater than T , exception handling is not performed (i.e., step (c) is bypassed).

In the following step, the hand and potentially compensated shoulder coordinates (based on threshold T) are transformed into world coordinates. As shown in Figure 3, the principal point of the world coordinates is located in the top-left position of the screen. The transformation is performed according to the following equations [15]:

$$x_k = \left(i - \frac{w}{2}\right) \times (z_k + \text{minDist}) \times \text{SF} \times \frac{w}{h}, \quad (1)$$

$$y_k = \left(j - \frac{h}{2}\right) \times (z_k + \text{minDist}) \times \text{SF}, \quad (2)$$

$$z = z_k + 4000, \quad (3)$$

where $\text{minDist} = -10$ and $\text{SF} = 0.0021$ are based on the calibration results of previous works [11] and i and j are the horizontal and vertical pixel positions of the captured image frame with a spatial resolution of 640×480 pixels. Because the default Z -distance value (z_k) can be as small as 400 mm, the Z -axis value in (3) must be compensated accordingly. Moreover, because the 3D coordinates (x_k, y_k, z) are measured from the principal point of the depth camera, the values of x_k and y_k should be adjusted by the offset ((D_x, D_y) in Figure 3) between the two principal world coordinate points and the depth camera coordinates. In our system configuration, D_x and D_y were 4450 mm and 950 mm, respectively, and were measured manually. The orientation variation between the Kinect and the screen is ignored. That is,

$$x = x_k + D_x, \quad y = y_k - D_y. \quad (4)$$

The two world coordinate positions for the shoulder and hand are given by (X_s, Y_s, Z_s) and (X_e, Y_e, Z_e) , respectively.

FIGURE 3: Geometric relation between the world coordinates and depth camera coordinates [10].

Next (step (e) in Figure 1), a 3D line equation is defined from these two 3D points using the following equation:

$$\frac{X_s - X}{X_s - X_e} = \frac{Y_s - Y}{Y_s - Y_e} = \frac{Z_s - Z}{Z_s - Z_e}. \quad (5)$$

Because the line equation is regarded as an arm-pointing vector and the planar equation of the screen is $z = 0$, the intersection point (X_i, Y_i) between the screen and the line equation is calculated in step (f) in Figure 1 as follows:

$$\begin{aligned} X_i &= -\frac{Z_e (X_s - X_e)}{Z_s - Z_e} + X_e, \\ Y_i &= -\frac{Z_e (Y_s - Y_e)}{Z_s - Z_e} + Y_e. \end{aligned} \quad (6)$$

The intersection point is the physical targeting position shown in Figure 4. Because the physical targeting position (X_i, Y_i) is given in millimeters, its position must be transformed into logical pixel coordinates (x_p, y_p) in order

FIGURE 4: Conceptual diagram for representing the pointing vector and targeting position [10].

to control the system mouse cursor position (step (g) of Figure 1). These logical pixel coordinates are given by

$$x_p = \frac{X_i \cdot x_{res}}{W}, \quad y_p = \frac{Y_i \cdot y_{res}}{H}, \quad (7)$$

where (x_{res}, y_{res}) is the spatial resolution of the screen and W and H are the actual width and height of the screen, respectively. For our system, $(x_{res}, y_{res}) = (1920, 1080)$, $W = 932$ mm, and $H = 525$ mm. Finally, the cursor position of the system mouse is moved to the calculated arm pointing position (x_p, y_p) using the WINAPI function `SetCursorPos(int x, int y)` [16].

2.2. Finger Counting Gesture Recognition. Finger counting gesture recognition is processed using the steps in the blue dotted box of Figure 1. In step (i), the right hand depth image is obtained based on the position of the right hand, which is acquired by using the Kinect SDK skeleton model. The spatial resolution of the image is 100×100 . The gray levels of the depth image indicate the Z-distance between the Kinect depth camera lens and the corresponding object. Therefore, the higher the gray level, the shorter the distance between the camera lens and the object. In order to extract the right hand's shape, the right hand depth image is binarized by regarding the higher gray level in the depth image as the threshold (step (j) in Figure 1). However, an outline of right hand shape that has been binarized only once will be articulated, as shown in Figure 5(a). An extracted edge from a once-binarized right hand image will contain bifurcation, which may disturb fingertip detection that uses edge tracking. To solve this problem, a once-binarized right hand image is blurred by using a 7×7 average filter, as shown in Figure 5(b). Subsequently, a binarization is performed again using the median gray value (128 in a 0–255 gray scale) to obtain a right-hand shape (step (k) in Figure 1). A hand shape image with a flattened outline can be acquired, as shown in Figure 5(c).

Then, hand outline detection must be performed, to facilitate fingertip detection. Assuming that the twice-binarized image and the structural element for morphological erosion ($\odot$) are A and B , respectively, the hand outline image ($\beta(A)$) can be extracted by subtracting the erosion image from A (step (l) in Figure 1) using the following equation:

$$\beta(A) = A - (A \odot B). \quad (8)$$

As a result, the outline image of the right hand can be acquired as shown in Figure 6.

Subsequently, counterclockwise edge tracking is performed; the edge pixel that has the minimum Y-axis value is used as the starting point. If two points on the edge have the same minimum Y-axis value, the point with the lowest X-axis value is used as the starting point. The 8-neighbor pixels (Figure 7(a)) surrounding the starting point are assigned priorities 1 through 8, as shown in Figure 7(b).

According to the priority, the 8-neighbor pixels are analyzed to determine whether the pixel is an edge (gray level value = 255) and whether it is “nonvisited.” If an edge pixel that is “nonvisited” is detected among the 8-neighbor pixels, the pixel is determined to be the new center position. Accordingly, the previous center position is marked as “visited.” These steps are repeated until no pixels are found that satisfy the two conditions (edge and nonvisited) among the 8-neighbor pixels.

If an 8-neighbor pixel priority is not assigned, edge tracking will be performed abnormally. For example, in the right hand edge of Figure 8, the minimum Y-axis value is determined as the starting point and is labeled in the figure. Edge tracking is performed by using the starting point as a center position. Then, the $(x - 1, y + 1)$ pixel of the starting point's 8-neighbor pixels is changed to the next center point, according to the predefined priority order. If the $(x + 1, y + 1)$ pixel has a higher priority than the $(x - 1, y + 1)$ pixel, the priority is appropriate for clockwise edge tracking. Therefore, 8-neighbor pixels that have a value of $(x - 1)$ as their X-index are assigned a higher priority than pixels that have $(x + 1)$ as their X-index, to facilitate counterclockwise edge tracking. Edge tracking proceeds normally until arriving at position A. In position A, if the $(x - 1, y + 1)$ pixel of the center point has a higher priority than the $(x, y + 1)$ pixel, the $(x, y + 1)$ pixel will not be visited. Then, in case the priority of the $(x + 1, y)$ pixel is higher than the $(x - 1, y)$ pixel, edge tracking will terminate abnormally when the bottom of A becomes the center position. Likewise, in position B, if the $(x - 1, y - 1)$ pixel has a higher priority than the $(x - 1, y)$ pixel, the $(x - 1, y)$ pixel will not be visited and edge tracking will terminate abnormally. To prevent these abnormal cases, edge tracking should be performed according to a predefined priority.

While edge tracking is performed, three sequential points, at fifth-next-adjacent intervals, must be extracted as shown in Figure 8 (red points). Then, the angle between the three extracted points as Figure 9 must be calculated, using the following equation (step (m) in Figure 1):

$$\theta = \left(\tan^{-1} \left(\frac{y_1 - c_y}{x_1 - c_x} \right) - \tan^{-1} \left(\frac{y_2 - c_y}{x_2 - c_x} \right) \right) * \frac{180}{\pi}. \quad (9)$$

Here, the angle of the three points is calculated using the `atan2` function included in `math.h` header of the C standard library [17]. However, the `atan2` function's output ranges are $-\pi$ to π . Therefore, if the value of $\tan^{-1}((y_2 - c_y)/(x_2 - c_x))$ is negative and the value of $\tan^{-1}((y_1 - c_y)/(x_1 - c_x))$ is positive, the opposite angle of the three points will be calculated, as shown in Figure 10(b).

FIGURE 5: Binarized right hand images; (a) once binarized image, (b) blurred image with 7 * 7 average filter, and (c) twice binarized image.

FIGURE 6: Examples of right hand outline images obtained by the proposed method.

To solve this problem, the angle of the three points is calculated using the following equation, as illustrated in Figure 10(c):

$$\theta = \left(\left(\tan^{-1} \left(\frac{y_2 - c_y}{x_2 - c_x} \right) - 2\pi \right) - \tan^{-1} \left(\frac{y_1 - c_y}{x_1 - c_x} \right) \right) * \frac{180}{\pi}. \quad (10)$$

Then, if θ is lower than the predefined threshold ($T = 110^\circ$), the center point of the three points is regarded as the fingertip (steps (n) and (o) in Figure 1). Finally, exception handling will be performed if one of the two noncenter points has already been identified as a fingertip, because if two of the three extracted points satisfy the condition, this indicates that the two points are on the same fingertip.

3. Experimental Result

To validate the proposed method, experiments were performed to measure the accuracy of the arm pointing and finger counting gesture recognition techniques. In the experiments, the distance between the subject's body and the screen was approximately 2.2 m. Software capable of recognizing upper body pointing gestures was implemented using C++, MFC (Microsoft Foundation Classes), and the Kinect SDK. The implemented software, as shown in Figure 11, could be operated in real time (approximately 18.5 frames/s) without frame delay or skipping on a PC with an Intel i7-3770 CPU, 8 GB RAM, and a 42-inch display.

In our first experiment, targeting accuracy for specific pointing positions was measured for eight subjects. Each subject pointed to five predefined reference positions (indicated

FIGURE 7: (a) 8-neighbor pixels and (b) assigned priority of the 8-neighbor pixels.

TABLE 1: Targeting error against reference positions [10].

Reference positions	Error without compensation			Error with compensation		
	X-axis	Y-axis	RMS	X-axis	Y-axis	RMS
1	62.90	52.19	81.73	16.95	22.04	27.81
2	4.54	5.95	7.49	4.29	3.41	5.48
3	6.33	6.95	9.40	0.54	5.87	5.89
4	2.4	0.54	2.51	14.04	7.25	15.80
5	7.79	3.25	8.44	10.12	0.91	10.16

Unit: pixel.

FIGURE 8: Edge tracking path example for explaining the necessity of the predefined priority (full arrow: normal edge tracking process, dotted arrow: abnormal edge tracking process).

by the “ $\times$ ” in Figure 12); this sequence was repeated three times. The indicated order was assigned randomly. Tests were

FIGURE 9: Example with three arbitrary points in a 2D coordinate.

performed with and without the self-occlusion compensation function in order to validate the performance of our proposed compensation method.

The measured accuracy results from the experiment are shown in Figure 12 and Table 1. Four outliers caused by detection errors of the hand or shoulder were not included. As shown in Figure 12 and Table 1, position 1 experienced

FIGURE 10: Angle calculation using three sequential points of the hand's outline edge. (a) Obtaining angle in a normal case. (b) Obtaining the opposite angle in error case. (c) Compensating error case (θ : calculated angle of three points at the intervals of five adjacent edge points).

FIGURE 11: Lab-made gesture recognition software based on the upper body skeleton model in the Kinect SDK. (a) Pointing gesture recognition software (white dot: shoulder point, yellow dot: hand point) [10]. (b) Finger counting gesture recognition software.

FIGURE 12: Experimental results of detecting target positions against reference positions [10].

a much higher error rate compared to the other reference positions. This can be attributed to self-occlusion occurring most frequently in position 1; specifically, both 3D shoulder and hand points are positioned on a single camera perspective line. After adopting the proposed compensation method, we confirmed improvements in targeting accuracy for position 1. In this case, the X-axis error received more compensation

TABLE 2: Accuracy of fingertip recognition.

Number of fingertips	0	1	2	3	4	5	Average
Accuracy of recognition	98	99	98	97	100	98	98.3

Unit: %.

than that of the Y-axis, as shown in Table 1. The average RMS errors from tests with and without self-occlusion compensation were approximately 21.91 pixels and 13.03 pixels, respectively.

In our second experiment, the accuracy of the finger counting gesture recognition method was evaluated to validate the fingertip detection method. Five subjects participated in the experiment. Each subject performed six predefined finger-counting gestures, regardless of hand orientation, as shown in Figure 13. The order of the finger gestures was randomly announced. The accuracy was measured by comparing the number of fingers in the hand gesture to the number of fingertips that were detected.

Experimental results from the accuracy measurement are listed in Table 2. Here, the accuracy of the three-finger gesture was lower, compared to the other finger counting gestures. As shown in Figure 14, the shape of the folded ring and little fingers in the three-finger gesture is sharper than

FIGURE 13: Examples of six different finger gestures used in the second experiment.

FIGURE 14: Shape comparison of folded ring and little fingers for one-, two-, and three-finger gestures.

TABLE 3: Average processing times for arm pointing and finger counting gesture recognition.

	Arm pointing gesture recognition	Finger counting gesture recognition
Average processing time	6.1	0.5

Unit: ms.

that in the one- and two-finger gestures. In one- and two-finger gestures, the thumb suppresses the folded ring and little fingers. Because the sharper shape of the ring and little finger in the three-finger gesture can be misinterpreted as fingertips, the three-finger gesture may have been interpreted as a four- or five-finger gesture. As a result, the average fingertip recognition accuracy for the six predefined finger gestures was 98.3%.

As shown in Table 3, the processing times for arm pointing and finger counting gesture recognition were considerably fast: 6.1 ms and 0.5 ms, respectively. The skeleton

model detection time was not included in the calculated times. These experiments demonstrate that our proposed method can accurately recognize pointing and counting gestures in an efficient manner.

4. Conclusion

In this paper, we proposed a method for performing both pointing gesture and finger gesture recognition for large display environments, using a single Kinect device and a skeleton tracking model. To prevent self-occlusion, a compensation technique was designed to correct the shoulder position in cases of hand occlusion. In addition, finger counting gesture recognition was implemented based on the hand position depth image extracted from the end of the pointing vector. Experimental results showed that the pointing accuracy of a specific reference position significantly improved by adopting exception handling for self-occlusions. The average root mean square error was approximately 13 pixels for a 1920×1080 pixels screen resolution. Furthermore,

the accuracy of finger counting gesture recognition was 98.3%.

In future works, we will define effective manipulation commands for the detected finger counting gestures. Further, the proposed method will be applied to immersive virtual reality contents [18–20] as a natural user interface method for performing interactive navigation and manipulation.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (NIPA-2014-H0301-14-1021) supervised by the NIPA (National IT Industry Promotion Agency).

References

- [1] Y. Yamamoto, I. Yoda, and K. Sakaue, "Arm-pointing gesture interface using surrounded stereo cameras system," in *Proceedings of the 17th International Conference on Pattern Recognition (ICPR '04)*, vol. 4, pp. 965–970, Cambridge, UK, August 2004.
- [2] K. Nickel and R. Stiefelhagen, "Pointing gesture recognition based on 3D-tracking of face, hands and head orientation," in *Proceedings of the 5th International Conference on Multimodal Interfaces (ICMI '03)*, pp. 140–146, Vancouver, Canada, November 2003.
- [3] K. Nickel and R. Stiefelhagen, "Real-time recognition of 3D-pointing gestures for human-machine-interaction," in *Proceedings of the 25th DAGM Symposium*, vol. 2781 of *Lecture Notes in Computer Science*, pp. 557–565, Springer, Magdeburg, Germany, 2003.
- [4] M. Kolesnik and T. Kuleba, "Detecting, tracking, and interpretation of a pointing gesture by an overhead view camera," in *Pattern Recognition: 23rd DAGM Symposium Munich, Germany, September 12–14, 2001 Proceedings*, vol. 2191 of *Lecture Notes in Computer Science*, pp. 429–436, Springer, Heidelberg, Germany, 2001.
- [5] Tracking Users with Kinect Skeletal Tracking, <http://msdn.microsoft.com/en-us/library/jj131025.aspx>.
- [6] P. Jing and G. Yepeng, "Human-computer interaction using pointing gesture based on an adaptive virtual touch screen," *International Journal of Signal Processing, Image Processing*, vol. 6, no. 4, pp. 81–92, 2013.
- [7] Y. Guan and M. Zheng, "Real-time 3D pointing gesture recognition for natural HCI," in *Proceedings of the 7th World Congress on Intelligent Control and Automation (WCICA '08)*, pp. 2433–2436, Chongqing, China, June 2008.
- [8] S. Carbini, J. E. Viallet, and O. Bernier, "Pointing gesture visual recognition for large display," in *International Workshop on Visual Observation of Deictic Gestures*, pp. 27–32, 2004.
- [9] R. Kehl and L. van Gool, "Real-time pointing gesture recognition for an immersive environment," in *Proceedings of the 6th IEEE International Conference on Automatic Face and Gesture Recognition (FGR '04)*, pp. 577–582, Seoul, Republic of Korea, May 2004.
- [10] H. Kim, Y. Kim, D. Ko, J. Kim, and E. Lee, "Pointing gesture interface for large display environments based on the kinect skeleton model," in *Future Information Technology*, vol. 309 of *Lecture Notes in Electrical Engineering*, pp. 509–514, 2014.
- [11] H. Park, J. Choi, J. Park, and K. Moon, "A study on hand region detection for kinect-based hand shape recognition," *The Korean Society of Broadcast Engineers*, vol. 18, no. 3, pp. 393–400, 2013.
- [12] J. Choi, H. Park, and J.-I. Park, "Hand shape recognition using distance transform and shape decomposition," in *Proceedings of the 18th IEEE International Conference on Image Processing (ICIP '11)*, pp. 3605–3608, Brussels, Belgium, September 2011.
- [13] I. Oikonomidis, N. Kyriazis, and A. Argyros, "Markerless and efficient 26-DOF hand pose recovery," in *Proceedings of the 10th Asian Conference on Computer Vision*, pp. 744–757, 2010.
- [14] C. Cao, Y. Sun, R. Li, and L. Chen, "Hand posture recognition via joint feature sparse representation," *Optical Engineering*, vol. 50, no. 12, Article ID 127210, 10 pages, 2011.
- [15] <https://groups.google.com/forum/#!topic/openkinect/ihfBIY561s>.
- [16] <http://msdn.microsoft.com/en-us/library/windows/desktop/ms648394%28v=vs.85%29.aspx>.
- [17] <http://msdn.microsoft.com/en-us/library/windows/desktop/bb509575%28v=vs.85%29.aspx>.
- [18] C. Ng, J. Fam, G. Ee, and N. Noordin, "Finger triggered virtual musical instruments," *Journal of Convergence*, vol. 4, no. 1, pp. 39–46, 2013.
- [19] J. McNaul, J. Augusto, M. Mulvenna, and P. McCullagh, "Flexible context aware interface for ambient assisted living," *Human-Centric Computing and Information Sciences*, vol. 4, no. 1, pp. 1–41, 2014.
- [20] A. Berena, S. Chunwijitra, H. Okada, and H. Ueno, "Shared virtual presentation board for e-Meeting in higher education on the WebELS platform," *Journal of Human-centric Computing and Information Sciences*, vol. 3, no. 6, pp. 1–17, 2013.