On attaining user-friendly hand gesture interfaces to control existing GUIs

Egemen ERTUGRUL¹, Ping LI², Bin SHENG¹

¹. Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China
². Faculty of Information Technology, Macau University of Science and Technology, Macau, China

* Corresponding author, egertu@sjtu.edu.cn
Received: 20 October 2019 Accepted: 26 February 2020

Citation: Egemen ERTUGRUL, Ping LI, Bin SHENG. On attaining user-friendly hand gesture interfaces to control existing GUIs. Virtual Reality & Intelligent Hardware, 2020, 2(2): 153—161
DOI: 10.1016/j.vrih.2020.02.001

Abstract Background Hand gesture interfaces are dedicated programs that principally perform hand tracking and hand gesture prediction to provide alternative controls and interaction methods. They take advantage of one of the most natural ways of interaction and communication, proposing novel input and showing great potential in the field of the human-computer interaction. Developing a flexible and rich hand gesture interface is known to be a time-consuming and arduous task. Previously published studies have demonstrated the significance of the finite-state-machine (FSM) approach when mapping detected gestures to GUI actions. Methods In our hand gesture interface, we broadened the FSM approach by utilizing gesture-specific attributes, such as distance between hands, distance from the camera, and time of occurrences, to enable users to perform unique GUI actions. These attributes are obtained from hand gestures detected by the RealSense SDK employed in our hand gesture interface. By means of these gesture-specific attributes, users can activate static gestures and perform them as dynamic gestures. We also provided supplementary features to enhance the efficiency, convenience, and user-friendliness of our hand gesture interface. Moreover, we developed a complementary application for recording hand gestures by capturing hand keypoints in depth and color images to facilitate the generation of hand gesture datasets. Results We conducted a small-scale user survey with fifteen subjects to test and evaluate our hand gesture interface. Anonymous feedback obtained from the users indicates that our hand gesture interface is adequately facile and self-explanatory to use. In addition, we received constructive feedback about minor flaws regarding the responsiveness of the interface. Conclusions We proposed a hand gesture interface along with key concepts to attain user-friendliness and effectiveness in the control of existing GUIs.

Keywords Human-computer interaction; Gesture recognition; Computer vision applications

1 Introduction

Hand gestures undoubtedly establish one of the most natural ways of interaction and communication in any environment, whether they are used for performing actions or interpersonal communication. At times, non-verbal communication is known to be more effective than verbal communication, especially when taking advantage of hand gestures. These facts are found to be useful and promising for human-computer interactions.
interaction (HCI). The search for new input techniques in existing user interfaces of personal computers and mobile devices, as well as recent studies on virtual reality and augmented reality\textsuperscript{11-18} point to the current interest in the use of hand poses and gestures in graphical user interfaces (GUIs) and virtual environments.

Previously proposed approaches for bare-hand tracking predominantly utilize motion capturing sensors or cameras. The former technique is less accessible for users with general, due to its requirements for dedicated hardware (e.g., gloves equipped with multiple motion sensors). Camera-based approaches\textsuperscript{9,10} take full advantage of the availability of cameras, enabling numerous users with various objectives to test and develop their applications without requiring dedicated hardware. However, solely being in possession of ubiquitous hardware does not suffice. There are different requirements between the run-time and development phases of a gesture-based interface, which might be in conflict. A customizable and extensible hand gesture interface that suits its users' needs requires considerable effort and time during the iterations of the development phase. Further, low-level fine-tuning of hand gesture recognition would require knowledge and expertise in computer vision and machine learning. Krupka et al. addressed these issues by introducing a set of development tools that utilize a simple language for hand pose and gesture descriptions\textsuperscript{111}. The proposed development tools and hand gesture interfaces rely on the finite-state machine (FSM) model, which helps users to seamlessly map hand gestures to actions, thus tailoring the interface to their needs. We further develop this idea and introduce a hand gesture interface that utilizes not only the hand gestures, but also their distinct features.

In our approach, each hand gesture activates a series of actions that take advantage of gesture-specific attributes, such as relative distances between hands and the camera, time of occurrences, etc. Furthermore, we provide additional features that we believe are essential for a better user experience on a hand gesture interface, such as the viewport interface for delivering real-time visual feedback to the user, on-screen notifications for displaying alerts, countdown timers, actions for handling exceptional behavior, cursor smoothing, and screen resolution mapping. The contribution of this study is two-fold. The first is a hand gesture interface, which is the product of a six-month-long iterative development with constructive feedback from experts in the field. The second is a supplementary application for capturing hand joints and keypoints in depth and color images for generating hand gesture datasets effortlessly and instantaneously. We believe that our approach and findings are significant for future practices on attaining user-friendliness in hand gesture interfaces and HCI. In the remainder of this paper, we first explain the methodologies adopted in the proposed hand gesture interface, then discuss our results and user survey, and finally provide conclusions.

2 Methodology

In this section, we thoroughly explain the methodologies employed in our proposed interface.

2.1 System overview

Our system consists of three main layers: Hand Gesture Interface, Input Simulator, and Graphical User Interface (Figure 1), where GUI is the only pre-existing element that is directly associated with the operating system. Figure 2 demonstrates the flowchart of the proposed Hand Gesture Interface.

The first layer is the proposed Hand Gesture Interface, which is designed essentially for gaining control over existing GUIs in off-the-shelf operating systems (OS), Windows in particular. We employed the Intel RealSense SDK\textsuperscript{12} for capturing and processing the stream of images captured by depth and color cameras to access full-hand skeleton/joints and detect gestures of the user. In addition to our interface, we used an external library, called "Input Simulator"\textsuperscript{413}, which forms a fundamental bridge and layer between our interface and the final layer, i.e., the built-in GUI.
The Input Simulator is a wrapper library that executes native Win32 input commands that are compatible with Windows machines. We map the gestures detected by the Hand Gesture Interface (Layer 1) to keyboard or mouse input and simulate them via the Input Simulator (Layer 2). The control parameters of the employed input functions are as follows:

- **Mouse.MoveMouseTo**(X_pos, Y_pos): This function is called at each frame update. X_pos and Y_pos are coordinates in the screen space to move mouse cursor to. These values are obtained by mapping the center of the hand with respect the camera space to the screen space (see Subsection 2.3).
- **Mouse.LeftButtonClick(), Mouse.RightButtonClick():** These functions perform a one-time click event at the current location of the mouse cursor.
- **Mouse.LeftButtonDown():** Performs a click-hold event at the current location of the mouse cursor until **Mouse.LeftButtonUp()** function is called.
- **Mouse.LeftButtonUp():** Releases the click-hold event initiated by **Mouse.LeftButtonDown().**
- **Mouse.VerticalScroll**(Direction): Performs a scroll event. It is used for zoom in-out operations. Direction is set between 1 or -1, based on the desired zoom in/out operation performed by the user.
- **Keyboard.KeyDown**(KeyCode): Simulates a key press-and-hold effect until **Keyboard.KeyUp**(KeyCode) function is called on the same **KeyCode**. KeyCode parameter is generally set to CTRL key to perform zoom in/out operation.
2.2 Gesture-specific attributes

Gesture-specific attributes are useful gesture properties that are acquired when hand gestures are performed. The key advantages of using these properties are that they give users the ability to use any static hand gesture in the form of a dynamic gesture and enable additional series of actions. Our implementation consists of switching between the four states, as showed in Figure 3:

1. **BeginIdle**: The state before executing an action. An idle gesture (i.e., spread of fingers) is being performed.
2. **BeginAction**: The state when the user starts executing an action. This state consists of a non-idle gesture (i.e., fist, V-sign, pinch, etc.) and saves the gesture properties (i.e., coordinates of hand keypoints, distances between hands and camera, etc.) with respect to the camera space at the time of occurrence.
3. **EndAction**: The state when the user completes executing an action. The interface executes the mapped action (i.e., zooming, selecting, clicking) at each frame using the current and saved gesture properties until it reaches this state.
4. **EndIdle**: The state after executing an action. An idle gesture (i.e. spread fingers) is being performed.

One of the foremost examples is taking advantage of the distance between the hands, and their respective depths with respect to the camera. In our interface, we mapped these gestures to zooming operations. Figure 4 demonstrates zooming in and out using two hands. The fist gesture executes the mapped operation and spreading the fingers stops it. It should be noted that the relative distance between the hands at the time of occurrence of fist gesture is taken into account when performing zooming operations. Moreover, users can continuously switch between zooming operations without needing to spread fingers during transitions.

Another approach incorporates the V-sign and spread fingers gestures using one hand only. As in the aforementioned example, we place emphasis on the time of occurrence. The key difference in this approach is the use of depth features of the detected gestures (i.e., V-sign gesture). Figure 5 and Figure 6 show that depth tracking is initiated when the user performs the V-sign. Intuitively, this operation can be used to perform zooming operations as well.

User experience is directly linked to the efficiency and ease-of-use of the system. We mapped gestures to actions that do not cause confusion in the user. Furthermore, RealSense SDK has constraints on employing certain gestures at once, due to concerns regarding the detection accuracy. Hence, we chose the activator and inhibitor gestures in accordance with how natural they feel when they are performed and how accurately they are detected by the SDK.
2.3 Supplementary interface features

Supplementary interface features are additional components of the hand gesture interface that significantly complement the user experience. In the absence of these features, the hand gesture interface would be able to operate as usual; however, it would lack user-friendliness and ease-of-use.

To begin with, we included a viewport display on the user interface, which delivers continuous real-time visual feedback to the user (Figure 7). Without a viewport display, users are prone to have difficulties using the interface, especially when determining whether their hands are within the field of view (FOV) of the camera.

Furthermore, we included on-screen notifications for displaying alerts and significant events fired by the RealSense SDK. These alerts show real-time statuses, such as hand detection, tracking, calibration, in/out borders (FOV), and closeness/farness with respect to the camera, etc. Notifications are set to be displayed briefly (lasting 2 seconds) at the corners of the screen without distracting the users.

By means of countdown timers, our interface can prevent users from performing unintended actions. Essentially, we employed countdown timers for time-critical GUI actions that require repetition or continuity, such as the double-click mouse operation (i.e., fast clicking) or right mouse click (i.e., by holding a gesture for a preset duration).

Furthermore, they serve to avoid using erroneous predictions returned by the SDK. For instance, the hand pinching gesture is mapped to the single left mouse click event (click and release), and the fist gesture is mapped to the continuous left mouse click event (click and hold). These two gestures are known to be frequently confused and misinterpreted by the SDK. Thus, running countdown timers with short durations after each gesture helps the interface compensate for such errors caused by the SDK.

The artificial joint position, called "JOINT_CENTER", Figure 8 is used as the center position of a hand. We mapped this position, tracked by the SDK, to control the cursor. The user chooses the hand (left or right side) to use as the cursor. This is a straightforward process, however, the center joints are generally inaccurate when directly mapped to the screen resolution of the existing desktop environment. Therefore, we perform linear interpolation at each frame between the current position of the cursor and the newly mapped position to achieve better accuracy and smoother transitions during cursor control. Moreover, the interface handles exceptional behaviors of its users, such as performing gestures when the prediction confidence is low or the hand location is near/inside the borders.
The confidence levels dramatically decrease when tracking hands across the borders of the camera's FOV. For that reason, it becomes challenging when the user decides to perform a clicking operation around the corners of the desktop screen. Instead of directly mapping the camera resolution to the desktop resolution (e.g., 640x480 depth resolution to 1920x1080 desktop resolution), we included offsets that help users effortlessly reach the corners of the mapped desktop resolution.

### 2.4 Gesture recorder

The gesture recorder seamlessly records the hand keypoints via color and depth images captured by RealSense SDK. Recorded hand features and images are essential for training custom hand gesture predictors and models. Moreover, it is important to carefully consider the anatomy of the human hand when recording the gestures. The software should be sufficiently tailored and generalized, such that it works for everyone who uses it. Figure 8 and Table 1 show the 22 joint keypoints that can be accurately accessed by RealSense SDK while segmenting the hand from the background and without requiring any calibration.

![Figure 7 Viewport display and user interface of hand gesture interface. The user is controlling a PDF reader application using both hands.](image)

![Figure 8 Joint numbers and positions are shown on a representative image of the left hand (left side) and connections between parent-child joints (right side).](image)

We aimed to make the gesture recording procedure as straightforward as possible. First, the user sets a gesture name and a time interval before recording the gesture. Once the recorder is started, the user...
performs the gesture, while the program records and saves the keypoint and image data at each tick of the timer. Thus, a shorter time interval will result in more data obtained. When finished, the user manually stops the recording. Lastly, the recordings are automatically saved in the corresponding folder, as in the examples shown in Figure 9. The directories of the first level are named after the gesture name entered by the user. If a directory with the same gesture name exists, the existing directory is reused. Subsequently, each new recording creates a new subdirectory under the corresponding gesture name, and a timestamp is given as its name. These recording subdirectories of gestures consist of a text file called "joint_data.txt" and depth images in the "bmp" format. The text file includes the time interval (TimeInterval, ms), recording number (RecordNo), body-side of hand (HandSide, Left/Right/Unknown), and positions of 22 joints (x,y,z) in meters. Each recording number matches the name of the depth image file.

### Table 1 Joint numbers, names, and corresponding parents

<table>
<thead>
<tr>
<th>Joint Number</th>
<th>Joint Name</th>
<th>Joint Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>JOINT_WRIST</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>JOINT_CENTER</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>JOINT_THUMB_BASE</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>JOINT_THUMB_JT1</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>JOINT_THUMB_JT2</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>JOINT_THUMB_TIP</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>JOINT_INDEX_BASE</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>JOINT_INDEX_JT1</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>JOINT_INDEX_JT2</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>JOINT_INDEX_TIP</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>JOINT_MIDDLE_BASE</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>JOINT_MIDDLE_JT1</td>
<td>11</td>
</tr>
<tr>
<td>13</td>
<td>JOINT_MIDDLE_JT2</td>
<td>12</td>
</tr>
<tr>
<td>14</td>
<td>JOINT_MIDDLE_TIP</td>
<td>13</td>
</tr>
<tr>
<td>15</td>
<td>JOINT_RING_BASE</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>JOINT_RING_JT1</td>
<td>15</td>
</tr>
<tr>
<td>17</td>
<td>JOINT_RING_JT2</td>
<td>16</td>
</tr>
<tr>
<td>18</td>
<td>JOINT_RING_TIP</td>
<td>17</td>
</tr>
<tr>
<td>19</td>
<td>JOINT_PINKY_BASE</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>JOINT_PINKY_JT1</td>
<td>19</td>
</tr>
<tr>
<td>21</td>
<td>JOINT_PINKY_JT2</td>
<td>20</td>
</tr>
<tr>
<td>22</td>
<td>JOINT_PINKY_TIP</td>
<td>21</td>
</tr>
</tbody>
</table>

Figure 9 Structure of recording directories and joint data.

## 3 Results and discussion

We tested our hand gesture interface on a notebook PC with Windows 10 OS, Intel Core i7-4700HQ CPU, at 2.40 GHz and 8 GB RAM. In addition to the existing GUI, we chose programs that are well-suited for general use to test our interface, such as the PDF document reader and presentation software. With the purpose of collecting responses on user experience, we conducted a user study on a small-scale with fifteen subjects from diverse backgrounds. Objective feedback from the users indicates that the gestures are sufficiently intuitive to effortlessly control the existing GUI and prevalent desktop applications. In contrast, most users reported discomfort caused by gesture recognition delays when switching between gestures and
performing actions that require responsiveness. We identified that these short delays are caused by the RealSense SDK, regardless of the hardware specifications, and they are most often noticeable during rapid transitions between different gestures. We mitigated the undesired outcomes of such delays, as mentioned in Subsection 2.3. Finally, users were generally satisfied with the offered range of actions. Figure 10 shows the results of our user study, indicating that the user experience was overall positive. Figure 11 shows a user experimenting with the proposed hand gesture interface.

It is important to recognize that our aim in this study is to propose an alternative user interface for controlling existing GUIs and not provide a replacement for conventional interfaces. We acknowledge that our approach may have shortcomings under certain circumstances, where it is more reasonable to use existing methods. However, there is no magic bullet when it comes to user experience. Different forms of user interfaces may provide various benefits in diverse environments.

4 Conclusions

We proposed a hand gesture interface for controlling existing desktop GUIs. Primarily, we emphasized user-friendliness and efficiency when developing the proposed interface. We described the essential notions applied to achieve these priorities throughout this paper. We believe that our findings will be valuable for future studies on hand gesture interfaces and in the HCI field. As part of a future study, we aim to improve the proposed interface to support a larger number of hand gestures with higher complexity, replace the RGB-D camera with an RGB-only camera, and adapt the presented techniques on mobile interfaces.

Acknowledgments: The authors would like to thank FreePik for gratuitously providing the vector images of hands that we used for our hand gesture illustration.

References


