Full-Body Gesture Recognition for Embodied Conversational Agents: The UTEP AGENT Gesture Tool

Ivan Gris, Adriana Camacho, David Novick

Department of Computer Science, The University of Texas at El Paso, El Paso, TX USA
{igris, accamacho2}@miners.utep.edu, novick@utep.edu

Abstract

Recognition of body gestures has long challenged developers of interfaces for real-time interaction between humans and embodied conversational agents (ECAs). In this paper we present a computationally simple approach to full-body gesture recognition along with an example of a human-agent application that makes use of it. We discuss how developers can use the tool to create pose libraries and how it works across different applications. And we evaluate gesture recognition implemented with the tool in the context of the human-agent application.

Index Terms: full-body gesture recognition, embodied conversational agents

1. Introduction

Full-body gesture recognition provides natural human-computer interaction in applications such as embodied conversational agents (ECAs). However, this approach to interaction remains difficult to achieve due to low recognition accuracy, distant sensor positioning, performance issues in real-time processing, intrusive interactive tracking technology, and the expense of capturing motion for representation of gestures.

For developers of ECAs, agent gestures can be animated or represented for purposes of recognition via hand-drawing or motion capture. For example, one agent with hand-drawn animation is a virtual nurse for hospital patients with low health literacy [1]. But hand-drawn animation is time-consuming and represents an artistic rather than naturalistic approach to gesture generation. And for recognition of gestures by human conversants, hand-drawn animation is highly problematic, in large part because each animation represents a particular movement path rather than a robust representation that accounts for variability in human motion.

Other ECAs use gestures generated via motion capture. [e.g., 2, 3]. This approach provides gestures that are more plausibly realistic, although it is certainly possible to capture and produce gestures that are idiosyncratic and unconvincing. Yet the motion-capture approach also can be time-consuming. For example, developing the relatively simple gestures for the ECA in the "Escape from the Castle of the Vampire King" game [4] took many weeks. And capturing gestures for purposes of recognition, which involves recording and processing the multiple examples of gestures needed for robust recognition, can require great effort.

To speed radically the process of capturing human gestures for purposes of generating ECAs' gestures and of recognizing the gestures of the ECAs' human conversational partners, we developed a tool that is capable of recognizing full-body gestures in real time and that can generate pose libraries for recognition across applications. In this paper, we review methods of gesture recognition that target different parts of the body, discussing the advantages and disadvantages of these methods. We present our gesture tool, explain how it works, and briefly describe the mathematical principles of full-body gesture recognition on which the tool is based, discuss the tool's potential applications. We discuss how we use the tool to aid with gesture annotation in real time and how the tool connects with our ECA system to enable real-time responses to gestures. We conclude with a discussion of the tool's limitations and how future updates will address these.

2. Background

To increase the believability and naturalness of human-agent interactions, developers seek to build agents capable of representing and interpreting traits that humans seem to do effortlessly. This includes the recognition of speech and gesture.

There are many commercial and research solutions to gesture recognition. Some target the face and focus on detecting emotions through facial features [5] or skin color [6], and others focus on gaze patterns. These systems, though, target specific body parts and usually require people to sit in front of a camera or sensor and maintain a relatively static position. With sensors like the Kinect, a device that is able to track user's body position and movements, users and developers alike have greater flexibility in terms of distance and gesture types. These sensors can be used at a short range to perform head [7], gaze [8] or hand tracking [9], while at greater distances they can cover the full body. This often involves a tradeoff, where detection at a short distance cannot be performed with a full-body setup, and vice-versa, leaving it to the developer's priorities to choose between full-body tracking versus head, gaze or hand tracking.

Although applications are often controlled through a computer screen and a traditional keyboard and mouse setup, some ECAs, such as those developed in our lab [e.g., 4] are life-sized projections of virtual human characters whose interaction instead aims for a more naturalistic approach using speech commands. The goal of these agents is to perform conversational tasks, often involving user-agent collaboration. To maintain the naturalness of the conversation, agents often need to react to the user's physical behavior, such as facial expression, gaze, and gesture, just like humans would. The more detailed the information about the user's non-verbal actions are, the better the agent can interpret and more accurately react to them [10, 11]. This enables a better interactive storytelling application, a domain of choice for full-body gesture recognition, as users can interact with objects contained in the same virtual space as the agent [12, 13].

These systems provide real-time full-body tracking in 3D, often including information about the hands and the face concurrently. However these systems can be costly and intrusive, meaning they often require users to wear special suits or markers to be detected by a set of several cameras positioned across an empty room. This sort of elaborate setup and its associated costs are not the only barriers to interaction

and gesture recognition. Even though they work with much greater accuracy, this information is usually processed and applied to 3D characters, meaning that the tracking information is translated directly into a virtual character to make the character replicate the actor's movements as closely as possible [14]. This means that there is no further analysis of the gesture-capture data, which makes impractical the identification of gestures and reactions in real-time to these gestures. And if a system does not identify full-body gestures automatically, this means that analysis of gestures will require, manual annotation of videos (e.g. [15]). Even though video annotation is nonintrusive and can be encoded on an abstract level, this is still a burdensome and time-consuming process.

To address these problems, we built a tool using Microsoft's Kinect sensor that suits specifically the full-body gesture recognition scenario while standing at a distance of six to eight feet away from the sensor. This tool is capable of generating poses for libraries that can be used for recognition through applications. Using this pose library, this tool identifies users' full body gestures in real time which enable the capability of analyzing the gestures performed by the user.

A similar system was recently developed that included similar functionality, although its current applications are game-oriented and is not actively maintained [16]. This system provided a full-body gesture recognition solution for existing applications, but this addressed only part of the challenge. When translating existing controls to gesture recognition, subjects are often required to perform the same gesture repetitively, and although the gestures can be metaphors of real-life gestures, they might not be ergonomic.

Accordingly, we designed our tool for detecting large sets of unique gestures and for users to create, export and import these gesture sets. Another key difference is that our tool can be used not only to interact with different applications but also to generate log files as spreadsheets that present the users' behavior across time, presumably facilitating researchers to analyze this data rather than the painful long process of annotating gestures manually.

Our approach sought to lower significantly the computational cost of gesture recognition. As discussed in Section 3, to make real-time recognition computationally feasible, our approach converts the 3D rendering to a 2D representation. An alternative approach involved using only depth information [17]. Again, to simplify recognition to reduce computation, our approach used a finite-state model for gesture recognition (see [18] for a review of alternative approaches generally, and see [19 and 20] for reviews of alternative approaches using the Kinect), although our approach is even simpler than the FSM model of [21] because it relies on pose sequences without timing information.

We connected our tool to a markup language and interpreter [22], a middleware system that enables external applications to access pose libraries and gesture detection. In addition, we created a user interface (see Figure 2) that enables developers to build pose libraries based on screenshots of the desired poses and that has additional features aimed at improving accuracy through basic statistical analyses. Figure 1 shows a human performing a "hi-five" gesture that is recognized and interpreted by an ECA.

3. Tool implementation

In this section we delve deeper into the implementation, features, and use cases of the UTEP AGENT gesture tool. The

Figure 1. *Human, interacting with ECA, performing a "hi-five" gesture. The human's gesture is sensed by a Kinect just in front of the projection wall and is interpreted via the gesture tool.*

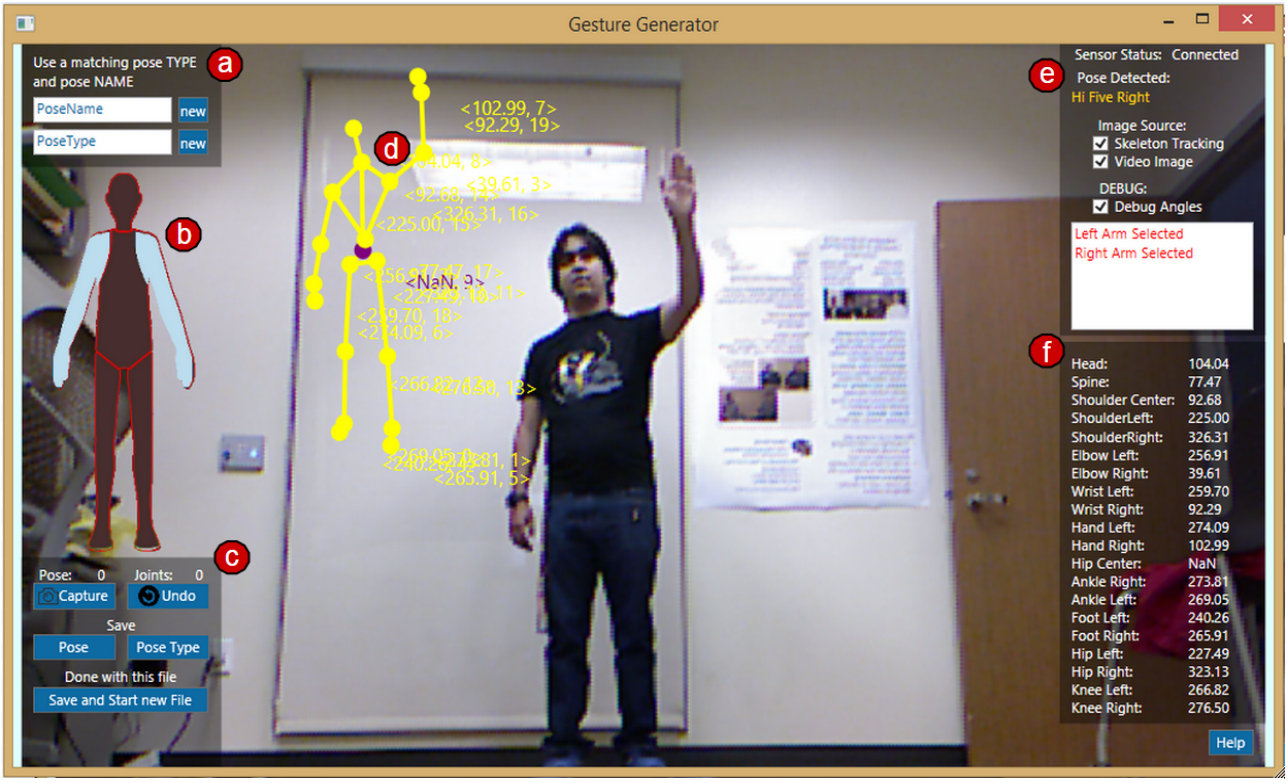

Figure 2. *UTEP AGENT gesture tool interface tracking a "hi-five" pose. (a) specify the name (e.g., high five) and type (e.g., right/left) of a pose; (b) selection of specific body parts for capture; (c) capture controls; (d) 2D rendering stick figure of a person with every dot representing each joint; (e) debugging tools showing the recognized pose (if any), record of your activity and turning section "f" (joint angles) on/off; (f) list of all 20 major joints recognized by the Kinect with its angle value.*

tool is built as a standalone Windows application that can be connected to Unity3D, a game engine that renders and animates our ECAs and the virtual environment in which these appear.

Based on depth information captured by the infrared camera from the Kinect, the tool renders a skeleton consisting of lines connecting 20 major joints of the human body (see Figure 2f). These skeleton is a 2D rendering in stick-figure style of the person recognized, as shown in Figure 2d, which shows a person performing a hi-five gesture like that shown in Figure 1. Although the Kinect is able to recognize human figures and track their joints in real time, it cannot differentiate between poses. In fact, the sensor produces only a visual representation of lines connecting the joints and updates them according to their position in 3D Cartesian coordinates. Microsoft's Kinect SDK [23] enabled pose detection.

The configuration of the subject's body joints, their position and posture, defines a pose. However, creating a pose recognizer from coordinates presents several problems during translation, rotation, and scaling of the skeletons. First, coordinates of the tracked joints change depending on the base position. That is, doing a pose while standing to the left of the screen will render different coordinates than doing the same pose on the right edge of the sensor's tracking field. This could be solved by using an offset parameter that checks for the same pose across different locations, but this approach can become computationally expensive, depending on the size of the pose library. A solution to this would be to calculate the offset based on an anchor point, in this case the hip joint, that controls translation, but this approach would remain ineffective for rotation and scale.

Scale is an issue, not because people come in all shapes and sizes but because they move. People do not grow and shrink in a few seconds, but they do change their distance from the sensor, which looks like a growing and shrinking effect to the sensor. In other words, when people translate along the z-axis, they appear larger or smaller on screen. This, combined with the x-axis translation, can make the process computationally expensive and unmanageable in real time.

To resolve these issues, first we eliminated the depth information. Because the rendering occurs in 2D regardless of the 3D information contained in the coordinates, and because in our research settings users are always located directly in front of the sensor at a relatively constant distance of eight to nine feet, the 3D information does little to help the gesture tool accuracy but does slow our system considerably.

Second, once the coordinates are transformed to 2D, each joint is triangulated using the parent joint (in this case the hip center) as base and creating a right triangle. We then use this triangulation to switch from location information to angles between joints to avoid normalizing position information in real time and to improve the accuracy of our measures and enable a more intuitive margin of error. Because positions are relative to the standing position of the person interacting with the tool, different coordinates could mean the same gesture, making it hard to classify or differentiate gestures that occur at a different standing position. By using angles, we can instead guarantee that they will remain constant regardless of the user's starting position. However a limitation still remains in our tool because it does not completely remove the ambiguity of angles. To address this ambiguity, pose capture and recognition have to be done in the same room with the same angle position of the Kinect performing the recognition.

The third step is to recognize a pose. However to do this there must already be information about the pose to be recognized. To address this, we created a pose library that contains an array of pose objects; Figure 3 presents sample code. These objects contain a subset of joint pairs and the

```
private void PopulatePoseLibrary(){
  this._PoseLibrary = new Pose[23];
  //…
  //Pose 2 - Lift Right hand as a Hi Five
  this._PoseLibrary[1] = new Pose();
  this._PoseLibrary[1].Title = "Hi Five";
  this._PoseLibrary[1].Angles = new PoseAngle[4];
  this._PoseLibrary[1].Angles[0] =
         new  PoseAngle(JointType.ShoulderCenter,
         JointType.ShoulderRight, 32, 20);
  this._PoseLibrary[1].Angles[1] =
         new PoseAngle(JointType.ShoulderRight,
         JointType.ElbowRight, 314, 20);
  this._PoseLibrary[1].Angles[2] =
         new PoseAngle(JointType.ElbowRight,
         JointType.WristRight, 266, 30);
  this._PoseLibrary[1].Angles[3] =
         new PoseAngle(JointType.WristRight,
         JointType.HandRight, 268, 30);
```

Figure 3. *Pose library sample code*

angles between them. For example, to recognize a "hi-five" pose we would be interested in the angles that form between the shoulder, the elbow, the wrist, and the hand joints. This is still not enough, however, because we need to mirror the values to enable gestures to be executed symmetrically for both left and right side of the subject's body.

Initially, this process required manually taking screenshots from the rendering of the angles of a person on screen performing the desired pose for integration into the library. These angles were then passed to Excel sheets and processed manually to calculate the average and a proper margin of error to populate the pose library. To automate this process, we created the gesture capture tool as a separate module. The capture tool enables the developer to select the relevant body parts for the pose capture, as shown in Figure 2b. Then it enables the capture of those joint angles, using the controls in Figure 2c. The process requires the developer to click a capture button while the pose actor is representing the pose in front of the sensor. In addition, the tool enables developers to capture several times the same pose from the same or different pose actors to improve accuracy. As different people do the same gestures differently, or the same person might slightly change posture between one attempt and another, the capture tool collects the data, analyzes it by calculating the maximum, minimum, and average angles, and estimates a range parting from the mean of the angles required to recognize these gestures in the majority of cases. This is effectively calculating a margin of error, depending on the variety of poses that were captured.

There is a tradeoff between multiple captures and few or single captures. The more captures of the same pose (or gesture) that are taken, the more accurate the recognizer becomes. But the increased precision may prevent users from being recognized properly due to the reduced margin of error. In contrast, smaller sets of training data might lead to over-coverage and large margins of error due to frequent outliers. To examine the results and identify these outliers we generate two files. One file contains the values of all captures for each joint angle, making it easier for us to find these outliers and, if necessary, to recalculate the margin of error. The second file is in xml format and contains tags for every joint identifier, its average angle, and its margin of error calculated by getting the smaller value of either the difference of the maximum and the average, or the minimum and the average.

When the gesture capture tool has defined the angles and their respective margin of error, the pose is then added to the pose recognition library and can now be named and detected. The resulting string of the detected pose can then be used to trigger events in other applications or simply collect the data of common gestures (e.g., hands on hips, arms crossed, hand on face). Once the poses are stored in the library, we can build gestures from them. Because gestures require movement, we define a gesture as a sequence of poses. Once a pose of the collection of poses that constitute a gesture is detected, the system then expects to detect a second pose (or some number of poses) that will integrate a gesture and only then be detected as such. In other words, when the user follows the pose sequence, the tool detects the gesture.

4. Evaluation

Currently, we are using the UTEP AGENT pose tool for several studies, including analysis of the amplitude or extraversion of gestures and poses. We also use the tool as part of an immersive jungle-survival application in which we evaluate the level of rapport between humans and virtual agents as a function of their non-verbal behaviors. These behaviors are recognized to enable physical interaction with the ECA and its virtual world.

For the jungle game application we defined two types of gestures in the pose library to be detected: task gestures and background gestures. Task gestures were performed where users had to accomplish a certain task (e.g., lift hand, strike, throw spear) to advance through the story. The background gestures were performed by the user but were not necessary to advance through the story (e.g., crossed arms, normal stance, hands in front, hand on shoulder, hand on face). At the same time, we automatically capture and annotate, in a log file, the background gestures so that we can avoid manually annotating hours of paralinguistic behaviors.

The annotation includes gestures from both the human and the ECA, because we know when the agent will change poses from the animations that are specified in scripted interaction. For the human, the gesture tool detects when the subject does a certain gesture and adds a corresponding time-stamped annotation. This results in a graph like the one in Figure 4, which shows the changes in gestures of both the agent and the user across time.

Each interaction session of the jungle game where these gestures were recorded lasted for about 40-60 minutes; we expected the user to perform eight task gestures to advance in the scripted story. Users were not instructed as to how to perform the gestures, which resulted in a longer period of people trying to figure out how to perform the gesture resulting in some variance of gesture performance. For

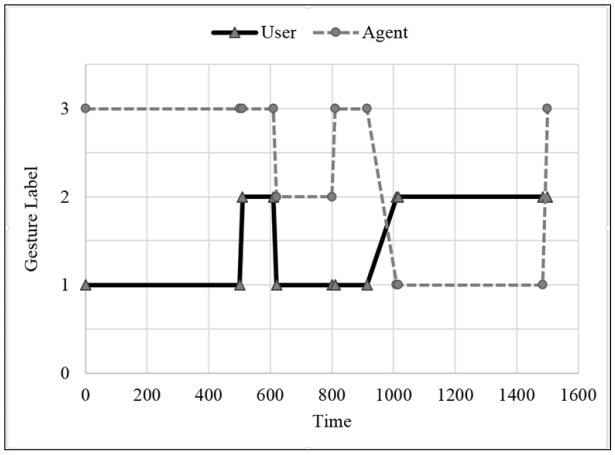

Figure 4. *User-agent gesture timeline. The numbers on the y-axis are labels of different gestures.*

example, they were asked to "strike a magnesium bar to light a fire." This gesture might not be as intuitive and does not have a standard way to be performed, which resulted low recognition rates for this gesture.

To evaluate the success/failure rate for recognition of the task gestures, we annotated the performance of these task gestures in the interaction of 30 users in the jungle game. The accuracy percentage was calculated with those task gestures that were performed correctly and recognized without a problem over the false negatives (when users performed the correct gesture and the gesture tool had a hard time recognizing them) and the false positives (when users did not perform the correct gesture and the gesture tool recognized them anyway) plus those gestures that were correctly recognized. For most task gestures, recognition accuracy ranged from about 50 to 80 percent; recognition was much lower for the unintuitive "strike 2" task. Table 1 reports these results. The recognition rates reflect multiple tries by the users; usually the users were able to achieve gesture recognition eventually.

Table 1. *Accuracy percentage of recognition of task gestures*

Gesture	Accuracy percentage
lift hand	70.73%
strike 1	51.02%
throw spear 1	49.02%
throw spear 2	65.79%
throw spear 3	70.27%
ventilate	77.78%
lift hand	65.85%
strike 2	25.00%

5. Discussion

Before the tool, generating a pose library took days or weeks of manually screening participants, getting their joint angle information, filtering the joints to remove the non-necessary joints for each poses, and collecting and aggregating the data from the different participants. With the tool, the process has been largely automated, and we now only need to have participants line up, stand in front of the sensor, and get scanned once per person per pose. For example, we used the tool to generate in less than an hour a library containing over 20 poses by scanning 12 members of our research group several times, with each person enacting a pose at a time. Participants did not receive any additional instructions apart from where to stand and what pose they had to enact. Each pose took about 15 seconds, and the 240 total poses from the 12 subjects took an hour to collect.

The tool, however, has several major limitations. Principally, aspects of the tool designed to reduce computational cost correspondingly eliminated consideration of information about depth and speed, our pose definitions average across all joints rather than focusing on the most meaningful joints, and our recognizer relies on context constraints to reduce confusion among gestures.

With respect to speed, the tool's gesture recognition in its current implementation is based on pose sequences that are insensitive to time. This means that a gesture will be recognized when the human follows a pose sequence regardless of the speed with which it is executed. This is not optimal, as gestures can vary in meaning depending on speed of execution [24]. We plan to integrate timers that can be set between poses to add greater precision to the gesture recognition.

We note, too, that the sequence of poses to detect a gesture can vary depending on users' performance and the accuracy of the Kinect in detecting a pose, making it difficult for a gesture to be recognized even if the user has performed the correct gesture.

In terms of joint angle accuracy, currently, our tool simply averages over different pre-labeled gesture instances and gives the developer the liberty to decide which body parts are relevant for a specific pose, treating all joint angles of selected body parts alike. Additional features, such as machine learning algorithms, could have been integrated for further refinement of the pose generation. A clustering approach, for example, could increase accuracy of pose generation by focusing only on relevant joint angles. In this case, a cluster of joint angles would represent a predefined pose in the tool. However, this approach would be limited by its inability to remove overlap within poses, an issue that is handled appropriately in our current implementation.

Another concern involves confusion among gestures. When gestures are not well defined, their margin of error might be higher than usual. If this happens across several gestures, there might be subsets of coordinates that fall between one or more gestures, making the recognizer unable to decide which gesture was actually executed. To avoid this, we activate and deactivate poses or gestures based on our expectations, much in the same way that we create contexts for speech recognition. By lowering the number of poses that can be recognized at the same time we decrease the overlap risk. This approach can be problem, however, when the user does not know what poses to expect or when two poses that overlap are expected. Moreover, our technique of reducing the joint positions to a 2D plane significantly increases the risk of confusion.

The tool has other limitations related to its implementation. Indeed, one of the tool's main advantages is also a disadvantage: it can perform all the data gathering and analysis in real time, but only in real time. This means that the tool cannot analyze a video recording after it has been captured. In contrast, motion-capture systems can store the 3D data and can be used at a later time for tweaking and post-processing to adjust for different physical traits among actors and the characters they represent. For some studies, we have been able to capture and store 3D information as rendered by the depth sensors of two Kinects. However, the data sets become large, making it infeasible to record several hours of conversation for further analysis or automated annotation. Moreover, the analysis cannot be executed in real time, and as it provides 3D depth data rather than a 2D skeletal representation, our tool cannot convert or interpret the data in these formats.

Although the tool is limited in terms of dimensional space and post-processing data handling, it has proven to be useful and reliable for our current applications. Provided that there is post-processing of the pose library to minimize overlap, the tool performs well even though it is a lightweight application in comparison to commercial motion-capture systems or other recognizers that are unable to process information in real time. As it is, with our ECA front-end applications, the tool can be applied to real-time interaction, real-time video annotation, and pose analysis.

In the future, we plan to implement the recognizer with 3D coordinates on a more powerful computer, to include timing information for gestures, and to update the recognizer and capture tool to work with the Kinect ONE, which offers greater accuracy and additional capabilities.

The UTEP AGENT gesture tool is available from the authors.

6. Acknowledgments

For their contributions to the studies through which we developed the gesture tool, we thank Diego A. Rivera, Mario Guitierrez, Alex Rayon, Laura Rodriguez, Paola Gallardo, Alfonso Peralta, Victoria Bravo, and Brynne Blaugrund.

7. References

[1] T. Bickmore et al., "Taking the time to care: empowering low health literacy hospital patients with virtual nurse agents," in *Proc. Conf. on Human Factors in Computing Systems*, Boston, MA, 2009, 1265-1274.

[2] M. Thiebaux et al., "Smartbody: Behavior realization for embodied conversational agents," in *Proc. 7th Intl. Joint Conf. Autonomous Agents and Multiagent Systems*, vol. 1, Richland, SC, 2008, 151-158.

[3] C. Bregler et al., "Turning to the masters: Motion capturing cartoons," *ACM Transactions on Graphics* 21, no. 3 (2002): 399-407.

[4] D. Novick and I. Gris, "Building rapport between human and ECA: A pilot study." In *Proc. HCI Intl*., Crete, Greece, 2014, 472-480.

[5] C. Busso et al., "Analysis of emotion recognition using facial expressions, speech and multimodal information," in *Proc. 6th Int. Conf. Multimodal Interfaces*, State College, PA, 2004, 205-211.

[6] G.A. Ramirez et al., "Color Analysis of Facial Skin: Detection of Emotional State," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conf. Computer Vision and Pattern Recognition*, Columbus, OH, 2014, 474-479.

[7] L.P. Morency et al., "Head gestures for perceptual interfaces: The role of context in improving recognition," *Artificial Intelligence* 171, no. 8 (2007): 568-585.

[8] D. Bohus and E. Horvitz. "Facilitating multiparty dialog with gaze, gesture, and speech," in *Int. Conf. Multimodal Interfaces and the Workshop on Machine Learning for Multimodal Interaction*, 2010, doi: 10.1145/1891903.1891910.

[9] P. Trigueiros et al., "Hand Gesture Recognition System Based in Computer Vision and Machine Learning," in *Developments in Medical Image Processing and Computational Vision*, Springer International Publishing, 2015, 355-377.

[10] J. Gratch et al., "Virtual rapport," in *Intelligent virtual agents*, pp. 14-27. Springer Berlin Heidelberg, 2006.

[11] J. Bailenson and N. Yee, "Digital chameleons automatic assimilation of nonverbal gestures in immersive virtual environments." *Psychological Science* 16, no. 10 (2005): 814-819.

[12] D. Thue et al., "Interactive storytelling: A player modelling approach," in *Proc. Artificial Intelligence for Interactive Digital Entertainment Conf*., 2007, 43-48.

[13] U. Spierling et al., "Setting the scene: playing digital director in interactive storytelling and creation." *Computers & Graphics* 26, no. 1 (2002): 31-44.

[14] E. Bevacqua, I. Stankovic, A. Maatallaoui, A. Nedelec and P. De Loor, 'Effects of Coupling in Human-virtual Agent Body Interaction', in *Intelligent Virtual Agents*, Boston, MA, 2014, pp. 54-63.

[15] M. Kipp, M. Neff and I. Albrecht, 'An annotation scheme for conversational gestures: how to economically capture timing and form', *Lang Resources & Evaluation*, vol. 41, no. 3-4, pp. 325-339, 2007.

[16] E.A. Suma et al., "Adapting user interfaces for gestural interaction with the flexible action and articulated skeleton toolkit." *Computers & Graphics* 37, no. 3 (2013): 193-201.

[17] K.K. Biswas and S.K. Basu, "Gesture recognition using Microsoft Kinect®" in *5th Intl Conf. on Automation, Robotics and Applications (ICARA*), 2011, 100-103.

[18] S. Mitra and A. Tinku, "Gesture recognition: A survey," in *IEEE Tran. on Systems, Man, and Cybernetics, Part C: Applications and Reviews,* 37.3 (2007), 311-324.

[19] O. Patsadu et al., "Human gesture recognition using Kinect camera," in *2012 Intl Joint Conf. on Computer Science and Software Engineering (JCSSE),* 2012, 28-32.

[20] J. Han et al., "Enhanced computer vision with Microsoft Kinect sensor: A review," in *IEEE Trans. on Cybernetics,* 43.5 (2013): 1318-1334.

[21] P. Hong et al., "Gesture modeling and recognition using finite state machines," in *Proc. 4th IEEE Int. Conf. Autom. Face Gesture Recogn*., Grenoble, France, Mar. 2000, 410–415.

[22] D. Novick et al., "A mark-up language and interpreter for interactive scenes for embodied conversational agents," in *Proc. HCI Intl. 2015*, Los Angeles, CA, in press.

[23] J. Webb and J. Ashley, *Beginning Kinect programming with the Microsoft Kinect SDK*. [New York]: Apress, 2012.

[24] M. Neff, N. Toothman, R. Bowmani, J. Fox Tree and M. Walker, 'Don't Scratch! Self-adaptors Reflect Emotional Stability', in *Intelligent Virtual Agents*, Reykjavik, Iceland, 2015, pp. 398-411.