Conversational Agent Module for French Sign Language using Kinect Sensor

Thomas Poulet*, Victor Haffreingue*, Taha Ridene**

*EFREI,
{thomas.poulet,victor.haffreingue}@efrei.net
**U2IS, ENSTA ParisTech
taha.ridene@ensta-paristech.fr

Abstract. Inside a CAVE different AR/VR scenarios can be constructed. Some scenarios use conversational agent interaction. In case of "deaf-mute" person the interaction must be based on sign language. The idea of this paper is to propose a "deaf-mute conversational agent" module based on sign language interaction. This innovative AR module is based on *Kinect* acquisition and real time 3D gesture recognition techniques.

Keywords: *Kinect* Camera, Conversational Agent, Human Gesture Recognition, 3D Motion Trajectory, Real Time Processing, Sign Language.

1 Introduction

In the past years, virtual reality have been subject to an exponential growth. An ever-increasing number of research have been conducted in various fields, covering military applications [1, 2], health care [3–5], sport [6, 7], teaching and education [8, 9], etc. More details about virtual reality applications can be found in [10–12].

In all applicative domains, virtual reality relies on the processing of the user actions. This is called the Interaction Cycle (IC), and it is divided into four main components, as we can see in the figure 1.

Inside a CAVE (Le SAS) (see Fig 2) we are producing different scenarios based on conversational agent interaction, for example "permis piéton" [13]. But, when the person is "deaf-mute" the interactions have to be designed around sign language, and a existing agents might not be suited, thus a new conversational agent must be developed.

We proposed in [14, 15] a *multi-Kinect* module for the tracking step in IC (see Fig 1). The idea of this paper is to use this *Kinect* cameras for both modules: "the tracking" and "the conversational agent based on sign language".

The next section briefly presents a state of the art of conversational agents inside virtual reality environments, sign language recognition, and the relation between the two. The main part, which covers the 3D gestural recognition principles, the real time gestural segmentation and comparison and the processing flow, is to be found under section 3. Section 4 is about our particular use case, French Sign Language recognition. Finally we will conclude with a summary in section 5.

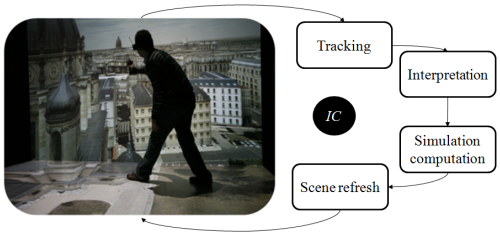


Fig. 1: Interaction Cycle – (3D Virtual City Visit – project TerraDynamica). Tracking: hands/head are tracked. Interpretation: gesture/position interpretation to deduce the camera position or the virtual activity. Simulation computation: the scene geometry is updated according to the tracked gesture interpretation. Scene refresh: update of the entire scene with the treatment result.

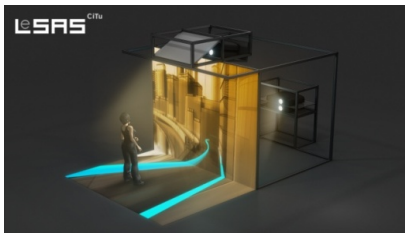


Fig. 2: "Le SAS": an immersive room with a two screens: wall and floor.

2 Conversational agent and Sign Language

Conversational agent[1] (CA) is a main module in some AR/VR applications. For example in "permis piéton" [13] we used Davi's[2] conversational agent platform. In [16] we can find language learning method with interactive virtual agent scenarios and speech recognition. A survey of conversational agent libraries can be found in [17, 18]. In case of "deaf-mute" persons, sensors like depth cameras or *Kinect* can be used to design this conversational agent.

Different research teams did some previous research on the use of *Kinect* for learning the hand and finger gestures. In [19] Wang and Lai use *Kinect* motion sensing capabilities to investigate peoples reactions and behavior based on their gestures. In [22] Chai et al. introduce a method to record, and recognize sign language gestures using a *Kinect* Camera. Chen et al. in [20] and Wang et al. in [21] use also the *Kinect* sensor as sign language translator.

Next sections are focused on our approach to 3D gestural recognition to produce a CA which will be used in AR/VR context.

[1] A dialog system or conversational agent (CA) is a computer system intended to converse with a human, with a coherent structure. Dialog systems have employed text, speech, graphics, haptics, gestures and other modes for communication on both the input and output channel. (Wikipedia definition)

[2] Davi : http://www.davi.ai/

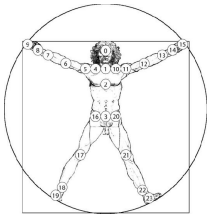
3 3D gesture recognition

The idea is to use 3D gesture recognition based on *Kinect* for learning a gesture library with their equity words, which will then allow us to recognize phrases for the immersive environment interaction.

3.1 System description

Our recognition and analysis system was developed using the *Kinect* Camera in association with the *FAAST* [23, 24] server. However, it is important to note that the described system is general enough to be used with any kind of sensors that can generate snapshots of elements' positions in three-dimensional space, one can for example think of *Leap Motion* or *Mocap System*.

Hardware architecture The system is composed of a *FAAST* server connected to a *Kinect* Camera. Each refresh cycle, the server generates and makes available the position of the joints. There are twenty-four joints which are located on different key points of the human body (see Fig 3). Every thirty three milliseconds, the *Kinect* camera makes an acquisition of the twenty-four joints in the 3D space and, via the *FAAST* server, sends it through the network. We can then make an analysis of the positions thirty times per second.



Sensor	*Joint*	*Sensor*	*Joint*
0	Head	**12**	Right Elbow
1	Neck	**13**	Right Wrist
2	Torso	**14**	Right Hand
3	Waist	**15**	Right Fingertip
4	Left Collar	**16**	Left Hip
5	Left Shoulder	**17**	Left Knee
6	Left Elbow	**18**	Left Ankle
7	Left Wrist	**19**	Left Foot
8	Left Hand	**20**	Right Hip
9	Left Fingertip	**21**	Right Knee
10	Right Collar	**22**	Right Ankle
11	Right Shoulder	**23**	Right Foot

Fig. 3: Twenty-four joints located on different key points of the human body given by *FAAST* server.

In order to speed-up the development process we choose to use the *VRPN middleware*[24]. This tool allows us to prepare and unify the data coming from the *Kinect*. This library is also highly versatile, we can thus easily add new sensors to our system, using its standard types and functions. The hardware architecture is described in figure 4.

Software architecture Our library is built in two parts, one part for supervised learning and another for recognition and analysis of a given sequence against a knowledge base. During the supervised learning, an expert user executes the gestures in front of the sensor. These gestures can be narrowed, refined during

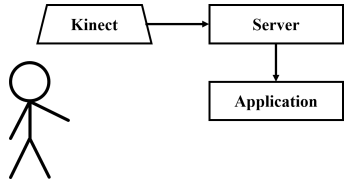


Fig. 4: Hardware architecture diagram presenting the different flow of interactions in our system. Our system, the server, interface itself with sensors (*Kinect*), and an application.

the recording session in order to speed-up future treatments. They are then combined into what we call grammar, in case of a sign language application. A grammar can contain all the gestures needed for a particular sentence.

During the recognition and analysis sequence, the non-expert user gestures are compared in real time to the knowledge base. During the comparison, the percentage of recognition of the gesture is displayed to the user. Our library can send this result as a real time stream of percentage, or in an interruption way which can inform the program of high detection score as they arrive. This particular case is useful when one wants to determine if the user did the gesture at a precise moment. We can see on the Figure 5 the position of our tool in the *Kinect* camera pipeline.

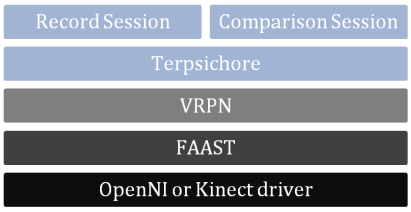


Fig. 5: Software stack presenting the different modules in our system. The blue modules are the elements of our solution. Our base framework (*Terpsichore*), aggregate data from different sensors before dispatching them to the recognition session or recording session module.

For our implementation, we chose to use a multi-threaded approach to be able to finely tune the efficiency of our algorithm. This architecture is not mandatory, during our testing we had not observed any major slowing by using only one thread.

Our implementation is running on at least two threads and three during a recognition session. One thread is pushing the information from the sensor and adding them to the queue. Another thread is working in parallel to extract the tendencies from the data in the queue. During the recognition session, a last one is comparing the tendencies.

3.2 Real Time movement segmentation

Problems and current solutions During our preliminary work we observed that the sensors used for recording often shares the same two characteristics, that can make real time recognition harder. They generate an important amount of data even though there is no movement, and they are highly prone to noise. The first one can be addressed by adjusting the sampling speed, but, this variation can impact directly the quality and thus the validity of the results.

A quick change in the movement could be completely ignored by the system. That being said, this method would not remove redundant points, for example during a steady state or a linear move. We would have a lot of redundancy that would not add any information to the comparison. The noise, result of the poor quality from used sensors, is extremely complex to remove while keeping data's validity, furthermore in a three-dimensional space. To exploit the data, we must have a pre-processing step, which will smooth the curve, using either Douglas-Peucker [25], Reumann-Witkam or other simplification algorithms [26].

However this approach may be really expensive in terms of computation, and thus incompatible with any real-time application. Moreover, the listed methods only apply on full curves, and cannot be used with real-time smoothing.

Our approach We have decided to apply pre-treatment to the acquisition of data to directly transform our series of points into a stream of motion vectors. This technique allows us to significantly reduce the data, while discarding the excessively noisy points (see Algo 1).

It is noteworthy that our method is not performing sampling for segmentation, thus it is completely independent of time variation. By not sampling we can guarantee that every movement as subtle as they can be would be detected. In addition, one can set the segmentation sensitivity on each axis, the system can thus eliminate variations on certain axis.

Algorithm The first step is to create a trend from the motion vector between the first two points, this trend will be the reference for further processing. When a new point is added, the motion vector between it and the last point of the latest trend is calculated, this vector is then compared to the last existing trend of motion vector. The comparison is performed using the calculation of vector rejection. This technique allows us to calculate a vector perpendicular to the base vector - the motion vector of the previous trend - and the end of the current motion vector. This Vector permits us to quantify the amount of shifting accomplished on each axis.

$$\overrightarrow{R} = \overrightarrow{a} - \frac{\overrightarrow{a} \,.\, \overrightarrow{b}}{\overrightarrow{b} \,.\, \overrightarrow{b}} * \overrightarrow{b} \tag{1}$$

In equation (1) R is the rejection of a from b.

It is noteworthy that the vector rejection follows a *Gaussian law* centered on π, the difference is maximum for an angle of $(2k+1)\pi$ and minimal for $2k\pi$. In the case of an angle of 4π, the result would be 0 and no change would be detected. However, we observed that this problem was not really one, resulting of our experiments with the sensor we have not managed to get a clear angle of 4π.

The vector thus obtained is then compared with the threshold, if it is greater, than it terminates the current trend and creates a new trend. This trend will be initialized with data from the current vector. Otherwise, the current trend tendency is computed by weighting its motion vector by the current vector one.

```
Data: raw data queue
current_data=Get data out of queue;
if tendency_list is empty then
    Create a new tendency out of the raw data;
else
    temporaray_tendency_vector = current_data - tendency_list.last();
    if tendency_list.last().global=0 then
        tendency_list.last().global = temporaray_tendency_vector ;
    else
        tendency_rejection =
        rejection(temporaray_tendency_vector,tendency_list.last().global);
        if tendency_rejection > limit_vector then
            Create a new tendency out of the raw data;
        else
            tendency_list.last().global = tendency_list.last().global*
            blending_factor+temporaray_tendency_vector *
            (1-blending_factor);
        end
    end
end
```

Algorithm 1: Algorithm of tendencies extraction.

Results In order to prove our system robustness, we tested it against five basic gestures (see Fig 6). Those are the building block of many more, the verification of their extraction is relevant.

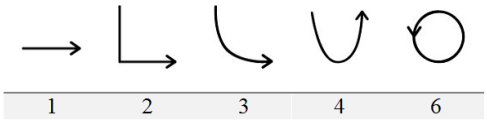
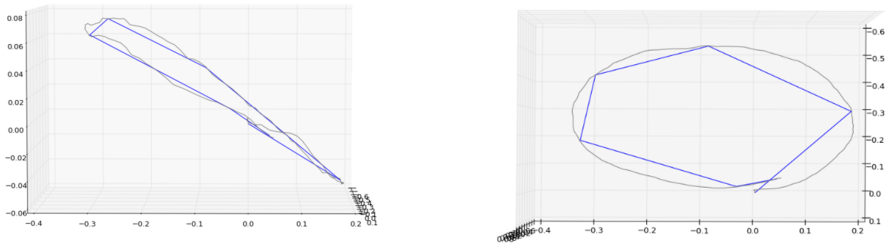


Fig. 6: Number of extracted vectors for each recorded gesture.

The moves are executed in two seconds for a total of 60 points.

We also conducted a visual check on the circle gesture. We can validate on the figure 7 the general appearance and significant noise reduction applied. The gesture was made in around four seconds for 125 points and the algorithm has reduced it to 6 trends.

The results confirm our first intuition, along with the method validity. Even if the reduction cannot be quantified in percent, because it is dependent of the gesture complexity, we can see a sensible reduction in the point numbers.



(a) X-Z (b) Y-Z

Fig. 7: Y-Z and X-Z view of the O movement. The gray lines are the original movement as recorded by the *Kinect*. The blue lines are our Tendency Extraction Algorithm's result.

3.3 Data structure and Library Storage

Internal Data Structures Our application is structured along the following types: the **grammar**, the **move**, the **tracker** and the **tendency**.

A grammar is a group of movements, linked in a context and/or by a logical sequence. A grammar is made of moves.

A move is a list of trackers, currently saved as a key/value structure to ease the access. The key contains the tracker's id and the value a pointer to the tracker.

A tracker represents a joint (3D skeleton point) in the user body representation, this translate into memory by its id along with a list containing the associated tendencies. During a recording session, we use a queue to store the *Kinect* data. This allows us to use the *FIFO (First In First Out)* principle with the buffer and bear the slow-down of the tendency decomposition system.

A tendency is stored in memory with its norm, its execution speed, and its shifting vector. To enhance the computations, we store the start point and the end point, we do not currently export those two data to the final data file.

Export Data Format To improve the exchange of data, along with the environment synchronization, we choose to create a simple file format with all necessary information needed to describe a movement. The file is in *JSON*, which gives us the possibility to multiply the interoperability possibilities. We use in our workflow a real-time visualization tool to render all gestures. This tool is written in Python and work smoothly with our *JSON* data files. The file format contains for each tracker the tendency list, a tendency contains the time needed to perform the gesture, as well as the global move vector. In order to facilitate the read and recognition tasks, we also added the list of trackers. This list allows us to remove quickly trackers from the recording that don't impact the gesture recognition.

3.4 Real time Gesture Comparison

Introduction During the comparison phase, our objective is to recognize, in a real-time movement vectors flow, elements that match elements saved in the library. There are already existing methods, which work using statistics and/or analytic approaches. Often being particularly costly in three-dimensional space, they are not suitable for real-time analysis across a large knowledge base.

To meet our needs, we develop a technique to quickly eliminate some possibilities, while comparing data with maximum confidence. We choose to address this issue by comparing the angle between each segment of the movement and its parallel in the movement in library. Once quantified, this angle is weighted, allowing us to determine the percentage of correspondence with the original gesture.

Parallel Walk We implemented a method called parallel walk, inexpensive to compare two sets of vectors (see Algo 2). It may at first seem attractive to compare the moves on a time basis, this is however not recommended. Doing so would make the comparison too much time dependent. A simple change in the speed of execution would then make the detection impossible. We choose our segmentation based on the vector norm, since the norm describes the movement. As for now on, the "real-time sequence" will describe the "sequence" output from the sensor, the sequence in which we are trying to find matches, and sequence the library recorded serving as a model for comparison.

During the initial step, we first establish trends on how the comparison will be performed. A pointer positioned on the end of our real-time sequence will move backward. The end point of this pointer is then used as the starting point of our parallel operation.

Following this initialization phase, we can begin the process of comparison, at each iteration the cursor will move forward simultaneously along the two sets of patterns. Every time it will encounter a rupture -a change in direction - it will calculate the difference between the two trends. If they are of different sizes, the algorithm will adapt the trends size to compare similar normalized vectors.

This method also allows early removal, if the difference is too big in the first stage, it is highly unlikely that the two actions are similar. This early elimination allows, on huge gestures base, to save considerable amount of time.

```
Data: tracker1, tracker2
tracker1_norm = get_tracker1_norm();
tracker2_norm = get_tracker2_norm();
tracker1_cursor = tracker1.begin();
tracker2_cursor = tracker2.end();
if (tracker2_norm > tracker1_norm) then
    for (i ← tracker2.size(); comparison_frame < tracker1_norm;
    − − i) do
        comparison_frame + = tracker2[i].norm();
        tracker2_cursor − = tracker2.size() - i;
    end
end
tracker1_distance = 0
tracker2_distance = 0
while (tracker1_cursor < tracker1.end() && tracker2_cursor <
tracker2.end()) do
    score += GetScore (tracker1_cursor, tracker2_cursor);
    max_score += PI;
    if tracker1_norm - tracker1_distance <= tracker2_norm –
    tracker2_distance ) then
        tracker2_distance = tracker2_cursor.norm() -
        tracker1_cursor.norm();
        tracker1_cursor ++;
        tracker1_distance = 0;
    else
        tracker1_distance = tracker1_cursor.norm() -
        tracker2_cursor.norm();
        tracker2_cursor ++;
        tracker2_distance = 0;
    end
end
```

Algorithm 2: Comparison algorithm (with parallel walk).

Metrics Coupled with parallel walk, we have metrics to compute the value of the difference between two trends. This algorithm can be adapted according to the needs and the results expected. In the case of our study, we choose to quantify the difference by the angle between the two vectors weighted by its norm (see Fig 8). Weighting by the norm allows us to take into account the relative size of the trends, so a subtle movement does not count as much as a large movement in the final result.

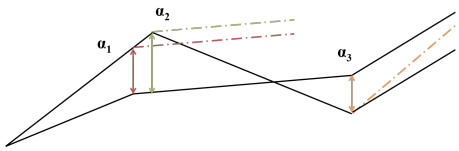


Fig. 8: Parallel walk representation. At each vector break, the angle α represents the difference between the two vectors.

We choose an angle approach to facilitate computations and speed up processing. In other use cases, an approach using areas may also be considered, depending on the expected accuracy, and computing power available. The angle computation is performed for each trend and for each tracker records. We can use metrics based on the following formula:

$$s^j = \sum_{i=0}^{n} cos^{-1}(\widehat{m_i^j}.\widehat{r_i^j}) * |v| \tag{2}$$

Therefore, we have the following equation for all tackers computation:

$$s_{final} = \sum_{j=0}^{o} S^j \tag{3}$$

With, “o” the tracker number, “n” the tendency number, “m” the tendency list from the model, “r” the tendency list from the record, and “v” the evaluated tendency norm.

Results Before applying our algorithm to our case of study we have verified the correctness of it with simple gestures used in section 3.2. To record the sequences, we executed each sign twenty times in front of the camera. We record the gestures by group of 10 to prevent any accustoming effect that may have biased our recordings. The detection result for each sign is recorded and placed in the following confusion matrix (see Fig 9).

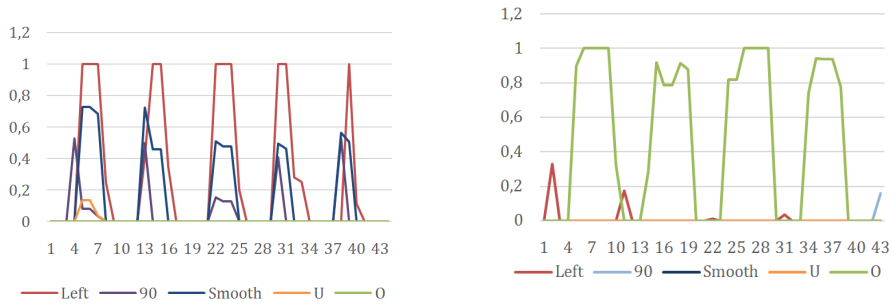
We immediately notice the relatively high level of confusion between the right angle and curve. Because of the segmentation process, there is a lot more matches of the curve when we are doing an angle than the other way. This is because we have to make a nearly perfect 90° angle to have a good probability of matching the angle. Thus, any angle variation would make the gesture recognized as a curve instead of a right angle.

We can observe higher amount of no detection for the U and O, this is related to the complexity of the gesture. The latter being composed of many trends, it is more complex to successfully reproduce without any visual help, but the results remain very satisfactory. We also set up a display of results over time.

	→	↳	↘	∪	○	no detection
→	100%	0	0	0	0	0
↳	0	65%	35%	0	0	0
↘	0	5%	95%	0	0	0
∪	0	0	0	75%	0	25%
○	0	0	0	0	90%	10%

Fig. 9: Confusion matrix between the five recorded gestures for twenty recording each.

The result included in figure 10a is an excerpt from a five-gesture recognition sequence to the left. The figure 10b presents the the result of a sequence extracted from a recognition sequence while matching four O gestures.



(a) (b)

Fig. 10: (a) Recognition percentages of five "left" gestures. (X: time scale; Y: percent of match between the current movement and the recorded). (b) Recognition percentages of four O gestures(X: time scale; Y: percent of match between the current movement and the recorded).

4 Application to French Sign Language (FSL)

4.1 Introduction

To field test our framework, we created an interaction system based on the French sign language (FSL) recognition. More and more systems use interfaces based on speech recognition to control applications. But for a "deaf", "partially deaf" or "mute person", the understanding of sign language is required.

4.2 Architecture

The system we currently use have the capacity to recognize a large number of gestures, we should be able, in the context of an augmented reality application with an artificial conversational entity, to record words and understand the gesture the user will do in front of the *Kinect*. Figure 11 includes the system global architecture.

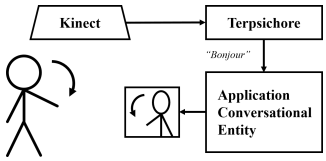


Fig. 11: Recognition Artificial conversational entity interaction architecture.

The gestures should be recorded before with an expert operator that understand and master the FSL.

The system will be built of a screen with the artificial conversational entity, which will talk with speech and sign language simultaneously. The screen will display also the detected sentence to enable the user to correct himself if the gesture has been wrongly recognized. The system will be completed with the *Kinect* sensor to record the user skeleton joints.

The user can now talk and freely interact with the application without the constraint of speech.

4.3 Context

The system is tested with the following sentence of French sign language:

J'ai acheté une voiture (I bought a car)

Figure 12 presents the outline of the sentence, translated in the French sign language.

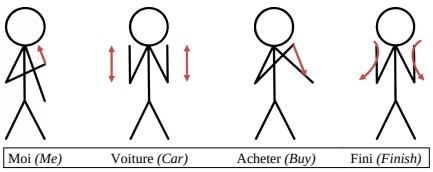


Fig. 12: French Sign Language sequence for the sentence "*J'ai acheté une voiture*" (I bought a car).

4.4 Result

We then try to match our sentence, which consists of four gestures. Each of them executed sequentially and recognized independently. We have the same process than before, except this time we do not repeat each word multiple time, but the whole sentence instead. To ensure the validity of our algorithm we execute this sequence several times in front of the camera. Figure 13 includes one of them.

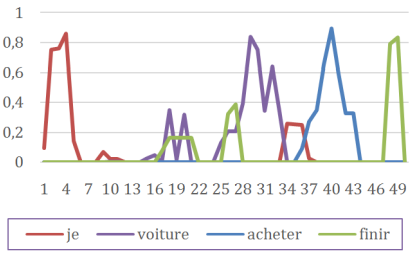


Fig. 13: Percentages of recognition for the FSL sentence (X: time scale; Y: percent of match between the current movement and the recorded)".

The threshold used for the recognition was of 0.8, we thus successfully recognized the sentence.

4.5 Analysis

The method we presented has some flaws for the sign language recognition usage. The current algorithm extract and compare vectors only movement-based, because the sign language also uses the start point of the movement, we could misunderstand some gestures. But, such a verification can be really simple to add in our methods.

The FSL use a lot of finger-based movements but the *Kinect* camera we used cannot record and track the fingers directly. Some other multi-sensors approaches can be used to refine the results, for example with the *Leap Motion* sensor [27], or with the *Kinect* camera information with a visual analysis of the hand [28, 29].

5 Conclusions and future work

In this paper we proposed a study of "deaf-mute conversation agent" module, tailored for sign language interaction. We proposed the detailed architecture (hardware and software) of this innovative AR module based on *Kinect* acquisition and real time 3D gesture recognition techniques. For gestural recognition we build a real time processing library and we exposed two algorithms about tendencies extraction and gesture comparison. Finally we tested our approach on real time French Sign Language recognition.

This module will be integrated in our interaction server architecture [30].

The next step will be to test our module inside a *CAVE* (see Fig 2), by extending AR/VR scenario *"permis piéton"*[13] for "deaf mute persons".

To include richer grammar, this FSL module will also be extended by fingers gesture recognition.

References

1. Baumann, J.: Military applications of virtual reality. Human Interface Technology Laboratory. www. hitl. washington. edu/scivw/EVE/II. G. Military. html (2010)
2. Rizzo, A., Parsons, T.D., Lange, B., Kenny, P., Buckwalter, J.G., Rothbaum, B., Difede, J., Frazier, J., Newman, B., Williams, J., et al.: Virtual reality goes to war: A brief review of the future of military behavioral healthcare. Journal of clinical psychology in medical settings **18**(2) (2011) 176–187
3. Riva, G.: Virtual reality in neuro-psycho-physiology: Cognitive, clinical and methodological issues in assessment and rehabilitation. Volume 44. IOS press (1997)
4. Laver, K., George, S., Thomas, S., Deutsch, J.E., Crotty, M.: Virtual reality for stroke rehabilitation. Stroke **43**(2) (2012) e20–e21
5. Zaal, F.T., Bootsma, R.J.: Virtual reality as a tool for the study of perception-action: The case of running to catch fly balls. Presence: Teleoperators and Virtual Environments **20**(1) (2011) 93–103
6. Bideau, B., Kulpa, R., Vignais, N., Brault, S., Multon, F., Craig, C.: Using virtual reality to analyze sports performance. IEEE Computer Graphics and Applications **30**(2) (2010) 14–21
7. Sanz, A., González, I., Castejón, A.J., Casado, J.L.: Using virtual reality in the teaching of manufacturing processes with material removal in cnc machine-tools. In: Materials Science Forum. Volume 692., Trans Tech Publ (2011) 112–119
8. Yair, Y., Mintz, R., Litvak, S.: 3d-virtual reality in science education: An implication for astronomy teaching. Journal of Computers in Mathematics and Science Teaching **20**(3) (2001) 293–306
9. Van Krevelen, D., Poelman, R.: A survey of augmented reality technologies, applications and limitations. International Journal of Virtual Reality **9**(2) (2010)
10. Zhao, Q.: A survey on virtual reality. Science in China Series F: Information Sciences **52**(3) (2009) 348–400
11. Fernando, T., Murray, N., Gautier, G., Mihindu, S., Loupos, K., Gravez, P., Hoffmann, H., Blondelle, J., Di Marca, S., Fontana, M., et al.: State-of-the-art in vr. Technical report, Technical report (2004)

12. Fuchs, P., Moreau, G., Guitton, P.: Virtual reality: concepts and technologies. CRC Press (2011)
13. Ridene, T., Leroy, L., Chendeb, S.: Innovative virtual reality application for road safety education of children in urban areas. In: International Symposium on Visual Computing, Springer (2015) 797–808
14. Salous, S., Ridene, T., Newton, J., Chendeb, S.: Study of geometric dispatching of four-kinect tracking module inside a cave. In: Proc. 10th International Conference on Disability, Virtual Reality and Associated Technologies. (2014) 369–372
15. Salous, S., Newton, J., Leroy, L., Chendeb, S.: Dynamic sensor selection based on joint data quality in the context of a multi-kinect module inside the cave" le sas". International Journal of Computer Theory and Engineering **8**(6) (2016) 471
16. Anderson, J.N., Davidson, N., Morton, H., Jack, M.A.: Language learning with interactive virtual agent scenarios and speech recognition: Lessons learned. Computer Animation and Virtual Worlds **19**(5) (2008) 605–619
17. Rubin, V.L., Chen, Y., Thorimbert, L.M.: Artificially intelligent conversational agents in libraries. Library Hi Tech **28**(4) (2010) 496–522
18. Van Lun, E.: Conversational agent. https://www.chatbots.org/conversational_agent/
19. Wang, H.C., Lai, C.T.: Kinect-taped communication: using motion sensing to study gesture use and similarity in face-to-face and computer-mediated brainstorming. In: Proceedings of the 32nd annual ACM conference on Human factors in computing systems, ACM (2014) 3205–3214
20. Chen, X., et al.: Kinect sign language translator expands communication possibilities. Microsoft Research (2013)
21. Wang, H., Chai, X., Zhou, Y., Chen, X.: Fast sign language recognition benefited from low rank approximation. In: Automatic Face and Gesture Recognition (FG), 2015 11th IEEE International Conference and Workshops on. Volume 1., IEEE (2015) 1–6
22. Chai, X., Li, G., Lin, Y., Xu, Z., Tang, Y., Chen, X., Zhou, M.: Sign language recognition and translation with kinect. In: IEEE Conf. on AFGR. (2013)
23. Suma, E.A., Krum, D.M., Lange, B., Koenig, S., Rizzo, A., Bolas, M.: Adapting user interfaces for gestural interaction with the flexible action and articulated skeleton toolkit. Computers & Graphics **37**(3) (2013) 193–201
24. Taylor II, R.M., Hudson, T.C., Seeger, A., Weber, H., Juliano, J., Helser, A.T.: Vrpn: a device-independent, network-transparent vr peripheral system. In: Proceedings of the ACM symposium on Virtual reality software and technology, ACM (2001) 55–61
25. Douglas, D.H., Peucker, T.K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Classics in Cartography: Reflections on Influential Articles from Cartographica (2011) 15–28
26. Agarwal, P.K., Har-Peled, S., Mustafa, N.H., Wang, Y.: Near-linear time approximation algorithms for curve simplification. Algorithmica **42**(3-4) (2005) 203–219
27. Potter, L.E., Araullo, J., Carter, L.: The leap motion controller: a view on sign language. In: Proceedings of the 25th Australian computer-human interaction conference: augmentation, application, innovation, collaboration, ACM (2013) 175–178
28. Li, Y.: Hand gesture recognition using kinect. In: 2012 IEEE International Conference on Computer Science and Automation Engineering, IEEE (2012) 196–199
29. Cooper, H., Ong, E.J., Pugeault, N., Bowden, R.: Sign language recognition using sub-units. Journal of Machine Learning Research **13**(Jul) (2012) 2205–2231
30. Ridene, T., Leroy, L., Chendeb, S.: Virtual Reality Server of Interaction eXtensible VRSIX. MVAR (Nov. 2016)