Intelligent Interactions: Artificial Intelligence and Motion Capture for Negotiation of Gestural Interactions

Quentin Thevenet, Marie Lefevre, Amélie Cordier, Mathieu Barnachon

Université Lyon 1, LIRIS, UMR5205, F-69622, France

Abstract. Gesture-based interfaces allow instinctive use of applications but are often limited by their arbitrary configuration. To overcome this problem, we propose to design adaptive systems able to negotiate new gestures with users. For that, we want to develop an assistance engine supporting the process of defining new gestures on the fly. The role of the assistance engine is to permit negotiation of gestures between users and the system. To facilitate the negotiation we propose to use Trace-Based Reasoning. In this article, we present a framework to collect and reuse traces in a gesture-based environment.

Keywords: Traces, Trace-Based Reasoning, Gestural interfaces, Motion Capture, Interactions.

1 Introduction

The democratization of motion capture devices (such as the Microsoft Kinect^{TM}) makes gestural interactions increasingly popular. We observe this mode of interaction mainly in video games, but other applicative areas are emerging. Most systems that implement gestural interactions define a predetermined set of actions available and expect users to perform these actions. For end-users, interacting with such applications can be very frustrating. Indeed, if they cannot perform a gesture required by the system or if the set of actions is incomplete with respect to their needs, end-users may not be able to achieve their goals.

To address these problems, we propose to develop a system of gestural interactions capable of learning while it is used. Our initial system is bootstraped with predefined gestures. We combine is with an assistance engine supporting the process of defining new gestures on the fly. The role of the assistance engine is to permit negotiation of gestures between users and the system. As a consequence, our final system is able to adapt itself to its users. To facilitate the negotiation we propose to use Trace-Based Reasoning. Traces help us to infer users' needs and to provide them with a user-friendly and relevant assistance. We perform this work as a part of the project IIBM[1] (Intelligent Interactions Based on Motion [1]), which combines researches in the fields of motion capture and artificial intelligence.

[1] http://liris.cnrs.fr/iibm/

In this paper, we focus on interaction traces. We show how we collect traces of gestural interactions and how we use these traces to assist users in their interactions with the system.

The paper is organized as follows. We first illustrate our motivations with a practical example in section 2. Section 3 presents a brief state-of-the-art regarding traces on the one hand and assistance to users on the other hand. Section 4 presents our approach and our framework to collect and reuse of gestural interaction traces. Section 5 discusses more specifically the use of trace-based reasoning to provide relevant assistance. Section 6 presents our implementation. A discussion is given in section 7.

2 Motivating scenario

To illustrate the context of our work, we present a simple scenario. In this scenario, we assume that a user is interacting with PowerPoint^{TM} by performing specific gestures. We make the assumption that we have a full environment enabling the user to do so. The system is able to recognize a set of predefined gestures and to associate them with specific actions within PowerPoint. Gestures are interpreted by a third-party software and are translated into instructions sent to PowerPoint (such as "Next slide").

When interacting with the system, the user may encounter several problems. These problems occur when movements are badly interpreted by the system. Causes of these problems are manifold. First, the gesture may be badly recognized by the capture system. This cause of failure is out of the scope of this paper. Next, the gesture may be badly interpreted. For example, the user moves his hand to the left, intending to perform a given action (e.g. "Next slide"), but the system performs another action. In this case, negotiation is needed to decide whether the system or the user is wrong. Another cause of failure is when the mapping between a gesture and an action is not available in the system. For example, a user might want to associate a wave gesture to the "Clear Screen" action. In this last case, negotiation is needed to enable the user to define new control gestures.

In this paper, we present a framework for collecting traces of gestural interactions. We show the mechanisms that exploit these traces to support the negotiation process between the user and the system. Our goal is to increase the adaptability of the system by supporting the creation of new gestures or the modification of existing gestures on the fly.

3 State-of-art

This section discusses the role of assistance in the design of adaptive systems. Then, it introduces the theoretical framework that we use to collect and exploit interaction traces.

3.1 Evolutive assistance

In a majority of research on assistance, assistance is defined as the system's ability to provide an answer to a problem given by the user. The role of the user is to provide information needed to find a solution [14]. This design of assistance is criticized because [2]: (i) it does not allow the user to acquire additional knowledge; (ii) it is contrary to the principle of practical assistance in a real situation (indeed, it is more useful to guide the user to find a solution rather than directly provide him with this solution [4]); and (iii) it does not allow the dialogue between human and machine that can guide and improve the search for solutions [10].

To overcome these limitations, an evolutive assistance must be proposed, *e.g.* an assistance able to adapt itself to the changing needs of users, assistance systems must be able to increase their knowledge over time [5]. According to [3], it is possible to use traces to propose an assistance adapted to user needs as the context evolves. Trace-Based Reasoning (TBR) [6] is an artificial intelligence paradigm similar to Case-Based Reasoning. It can solve new problems by reusing past experiences. In [5], the authors present an architecture for assistance TBR. This architecture relies on several knowledge bases that evolve during the use of the system. Reasoning mechanisms use these knowledge bases and thus improve their results over time. In our work, we use this principle to provide an assistance that can adapt itself to the user and evolve over time.

3.2 Interaction traces

Many studies focus on the production and exploitation of interaction traces. According to [11], a trace is as a set of observed elements temporally situated, called *obsels*. Obsels always have a timestamp. A trace model defines the structure and the *obsels* type that are contained in a trace, as well as the relationships between these *obsels*. A modeled trace, or M-Trace, is a trace associated with its trace model. There are two types of modeled traces.

Primary traces: the results of the *obsels* collection process. A primary trace of users actions may contain, for example, the *obsels*: "ctrl key", "c key", "ctrl key", "v key".

Transformed traces: traces that are produced from one or more source traces on which a transformation method is applied. A transformation method may for example be a temporal filter to keep only the *obsels* located in a given time interval. All traces can be transformed. For example, a transformation may convert the four previous *obsels* in a transformed *obsel* called "cut and paste".

A Trace-Based Management System (TBMS) is a system managing traces [8]. A TBMS has three main components: the collection module, responsible for storing *obsels* into traces, the transformation module applies transformations on traces, and the query module allows the manipulation of traces.

The assistance engine, presented in the proposed framework (see subsection 6.2) is built upon user's traces, collected during its used of the targeted application.

4 A framework using traces for negotiating gestural interactions

In this section we present our framework for the negotiation of gestural interactions. This framework contains several features: support of interaction between the user and the target application, gestures interpretation module, interactions tracing module, and assistance module. First, we discuss the knowledge models that are used in the framework. Next, we show how the various components are connected.

4.1 Knowledge models

The framework uses several knowledge models. The targeted application model allows us to know actions that the user can do in the application, and the context in which these actions are available. For example, this model indicates that, when one is in "presentation mode" with PowerPoint, the actions "next slide" and "previous slide" are available.

The traces model defines the types of *obsels* that are contained in traces. All obsels are timestamped. Traces record all the interactions (gestures and keyboard events). Our trace model contains the following *obsels*.

- **Gestural event**: informations about the position, direction and speed of movement of each part of the user's body in order to transcribe the gesture made.
- **Keyboard event**: key code and the status of the key (pressed or released).
- **Mouse event**: information about mouse movements and state of the buttons.
- **Targeted application event**: information describing actions performed on the targeted application and parameters of actions.
- **Assistance event**: contains information about the assistance provided to the user and his response.

4.2 General framework

To provide assistance to a system based on gestures users, we propose the framework in Figure 1. In this framework, a user interacts with an **application** (see 1) using gestures captured by a **motion capture** system (see 2) and / or a standard interface such as a keyboard or a mouse.

The different interactions of the user are processed by the **interpretation engine** (see 3). The interpretation engine converts gestures into actions understandable by the target application. To do this, it seeks gestures done in the

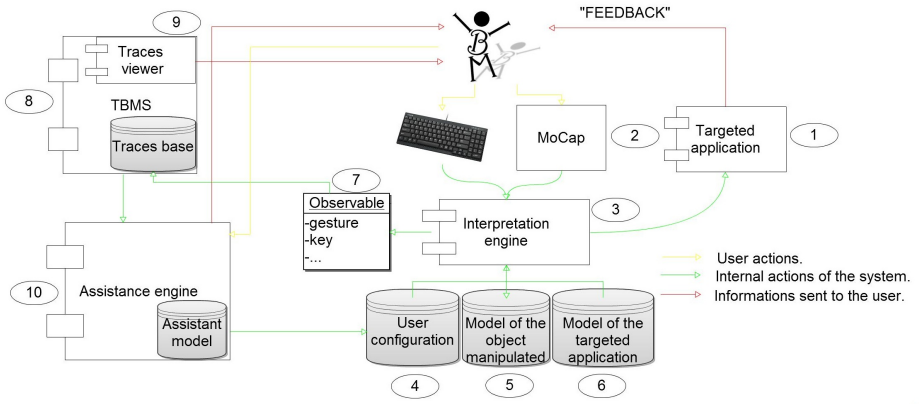

Fig. 1. Framework to use traces for negotiating gestural interactions.

configuration file of the user (see 4) to find the associated action on the target application. This module is also used to collect **observables** (see 7). The **model of the target application** (see 6) describes possible actions of the user on the application. The **model of the manipulated objects** (see 5) describes the current state of the target application. Based on these two models, the interpretation engine creates observables to describe events occurring on the target application. This module is also responsible for updating the model of the manipulated objects.

The **TBMS** (see 8) collects observables, creates *obsels* from these observables and builds primary traces by grouping them. It provides transformation mechanisms on traces. It also allows **visualization** of these traces (see 9) for the user. The **assistance engine** (see 10) searches, in traces, error situations. This module is the core of our assistance system and is described in detail in the next section.

5 Providing assistance based on gestural interaction traces

In this section we show how we can exploit gestural interaction traces to provide assistance to users of an interface based on gestures. It must be noted that the context in which is the user is important to decide if assistance must be provided or not. For example, if the user is doing a presentation in front of the audience, assistance must not interrupt him. It is then shut down. Deciding when to provide assistance is briefly discussed in section 5.2.

5.1 Using pattern recognition to identify assistance needs

First, we need to detect situations where user may need assistance. For that, we seek specific patterns in traces. These patterns detect failures during the use

of the system (e.g. something went wrong), that is why we use the term failure pattern. For now, we identified two failure patterns.

The first pattern is called *inconsequential gesture*. It occurs when the users performs several times the same gesture before pressing a keyboard key. For example (see Figure 2), the user moves several times his arm to the right, intending to go to the next slide. But nothing happens. Finally, the user gives up and uses the right arrow on the keyboard to perform his intended action.

Several types of assistance can be provided in this situation. First, the system can show to the user the proper gesture to perform the action. Next, the system can interact with the user to adapt itself to the user's needs. Indeed, if the user cannot remember the gesture associated to an action, or if he does not like it, or if the gesture does not exist yet in the system, he can define a new one.

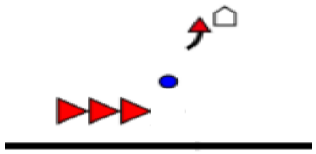

Fig. 2. This trace illustrates the pattern "inconsequential gesture"; red arrows correspond to the "moving arm to the left" gestures, blue round indicates a key pressed. The last arrow indicates that an action has been performed in the application (next slide), and the white pentagon shows that an assistance process has been triggered.

The second pattern we have identified is called *Action/Cancellation*. It corresponds to a succession of actions and cancellations. It occurs when the user performs unconsciously some gestures which cause unwanted actions. Consequently, they immediately cancel the action performed. In order to identify when a user cancels an action, we use the target application model. This model indicates, for each action, which is the reverse action. For example, it indicates that the reverse action of "next slide" is "previous slide".

Again, several types of assistance can be provided in this situation. Depending on the user's needs, we can offer him to change the mapping gesture / action either of the initial "action" gesture, or the "cancellation gesture", or even both.

Table 1 sums up the patterns we are able to identify for the moment and the various assistance possibilities we offer. In the following, we will show how trace-based reasoning will help us to dynamically identify more failure patterns.

5.2 Trace-Based Reasoning to improve assistance

Assistance presented in the previous section can be improved by using Trace-Based Reasoning (TBR) [6] to: identify the most appropriate assistance for a given failure situation, and identify the situations where the user needs assistance.

Failure	Assistance
Inconsequential gesture	show the gesture that enables the action; change the gesture that enables the action;
Action/Cancellation	change the gesture that makes the first action; change gesture that makes the second action; change both gestures;

Table 1. Pattern of failures situations and corresponding assistance.

When the assistance engine detects a pattern in interaction traces, it proposes to the user several assistances on the fly. The user may choose one option among them. For example, the user interacts with the system and the pattern X is detected. The system proposes several options of assistance A1,. . . ,An. Let's assume that the user chooses A2. If this situation is repeated several times, TBR can infer that the pattern X should be associated to assistance A2 by default. To give an actual example, we assume that the gesture for the action "next slide" is "move the right arm to the right", with a low amplitude. If the user moves a lot his arm while he speaks, he can switch to the next slide inadvertently. Therefore, he will cancel this unwanted action. TBR will detect a failure situation. Similar situations will be retrieved and reused to help the user. Here, the system will propose to the user to change the gesture amplitude for this action. The user is free to accept the modification or not.

If the user needs assistance, but is in a context that does not match any known pattern, the system cannot offer any support yet. In this case, TBR could exploit past experiences to provide relevant assistance. For that, the assistance engine will search contexts similar to the current one in traces. Then it will identify, in these contexts, the next actions to perform. The engine will adapt the actions according to the current context and recommend them to the user. For example, a user comes to a slide containing only a video. The reasoning system find in previous traces that usually, in this kind of slides, the users start immediately the video. The assistance system could therefore directly play the video and save the user an action. These two examples can be generalized by using a trace base of multiple users. By using TBR, it is possible to improve the assistance engine. TBR will allow us to discover new mappings between gestures and actions. In addition, it will allow us to discover new patterns of assistance.

6 Implementation

In this section, we present the framework we have implemented. We have used third-party tools for external tasks. The framework is implemented in C++. Up to now, we have implemented a gestural interface for PowerPoint and we have developed an assistance engine able to identify two failure patterns. We have experimented the system with ten different users but the results of this experimentation are not described in this paper.

6.1 Third-party tools

PowerPoint. In order to experiment our tool with all types of users, we chose to instrument a widespread tool. This is the reason why we decided to work with PowerPoint. Controlling PowerPoint with gestures is very intuitive. For example, we can move our hands to the left or right to move to the previous or next slide.

Kinect and FAAST. We decided to build on an existing tool to implement the motion capture component. The use of commonly depth cameras [12] was preferred to marker-based solutions, (like the Vicon system[2]) which need special equipments (expensive infra-red cameras and special suit), and are much more expensive. Our choice was motivated by the fact a device such as Kinect allows an immediate and instinctive interaction with the interface. Furthermore, it can be used in real-time [9], contrary to marker-based solutions. FAAST [13][3] is a free middleware, which allows integration of control gesture-based in the fields of video-games and virtual reality. FAAST emulates a keyboard by binding body posture and simple gestures to keys of a regular keyboard. Customized controls are defined in a configuration file (mapping between gestures and keys). Here, FAAST is configured to associate twelve gestures to keys. In order to avoid confusion with the actual keyboard, we mapped the gestures to keys that do not appear on a classical keyboard (F13 to F24).

Abstract Lite. Abstract Lite [7] is used to propose feedback to the user by showing him his own interaction trace. This graphical visualization enables the user to better understand how the system behaves.

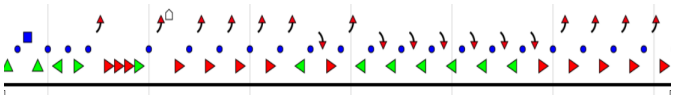

Fig. 3. Trace view with Abstract Lite.

Figure 3 shows an example of traces from our demonstrator in Abstract Lite. The blue square represents the beginning of the presentation. Triangles indicate gestures made by the arms. Arrows indicate transitions to next or previous slides.

6.2 Framework implementation

Interpretation engine. The Interpretation engine uses Windows API to collect keyboard and mouse events. It converts gestures into PowerPoint keyboard shortcuts according to the configuration file of the user. It uses a PowerPoint model and a current presentation model. The PowerPoint model allows to know the possible user's actions in the current state of presentation. The state of the presentation is saved in the model of the current presentation.

[2] http://www.vicon.com
[3] http://projects.ict.usc.edu/mxr/faast/

Trace-based management system. The TBMS transforms observables collected by the interpretation engine into *obsels*. *Obsels* are then stored in an XML trace. The trace-based management system allows us to perform requests on traces to exploit them.

Assistance engine. The assistance engine exploits traces to discover new knowledge. For the moment, it only implements a pattern recognition mechanism to discover failures in traces. Moreover, it offers a list of assistances when one these patterns is detected. We have implemented two patterns: the "Inconsequential Gesture" pattern and the "Action/Cancellation" pattern.

7 Discussion and conclusion

In this paper, we presented a generic system to assist and optimize the use of a gestural interface. To enable this support, all user's actions with the system are traced. Collected traces allow the system to identify the context in which the user is in, and to detect if assistance is needed. We have developed a framework to experiment with gesture-based interfaces. This framework contains a target application, a gesture interpretation module, a trace-base management system, and an assistance module. So far, we have implemented a first level of assistance based on traces. However, this type of assistance remains static. It is limited to the identification of predefined failure situations.

Therefore, it is necessary to develop mechanisms to enhance assistance on the fly. We propose to implement trace-based mechanisms, as suggested in [6]. A first idea is to reason on traces to automatically detect situations in which assistance should be provided. A second idea is to look for patterns in traces and to exploit these patterns to improve the usability of the system. For example, the system can identify that a user has to perform several gestures to perform a single action. In this case, a negotiation process can be triggered in order to map a new gesture (chosen by the user) to this action. More generally, many assistance scenarios, traces-based, can be imagined. Traces can be used to capitalize on the experiences of other users and to reuse these experiences.

When providing assistance to users, the main problem is to ensure that assistance does not disturb the user activity. Therefore it is necessary to find ways to trigger assistance timely, on purpose. Moreover, we need to consider the differences between users. The main perspective of this work is to explore ways of providing assistance to users, in the most pleasant way possible.

References

1. Barnachon, M., Ceccaroli, M., Cordier, A., Guillou, E., Lefevre, M.: Intelligent Interactions Based on Motion. In: Belén Diaz-Agudo, A.C. (ed.) Workshop CBR and Games, ICCBR 2011 (Sep 2011)
2. Cahour, B., Falzon, P.: Assistance à l'opérateur et modélisation de sa compétence. Intellectica 12(2), 159–186 (1991)

3. Champin, P.A.: ARDECO: an assistant for experience reuse in Computer Aided Design. workshop From structured cases to unstructured problem solving episodes - WS 5 of ICCBR'03, Trondheim (NO) pp. 287–294 (2003)
4. Coombs, M., Alty, J.: Expert systems: an alternative paradigm. International Journal of Man-Machine Studies 20, 21–43 (1984)
5. Cordier, A., Lefevre, M., Jean-Daubias, S., Guin, N.: Concevoir des assistants intelligents pour des applications fortement orientees connaissances : problématiques, enjeux et étude de cas. In: Despres, S. (ed.) IC 2010 - 21emes Journées Francophones d'Ingenierie des Connaissances. pp. 119–130. Presses des Mines (Jun 2010)
6. Cordier, A., Mascret, B., Mille, A.: Extending Case-Based Reasoning with Traces. Tech. Rep. RR-LIRIS-2009-005, LIRIS UMR 5205 CNRS/INSA de Lyon/Université Claude Bernard Lyon 1/Université Lumière Lyon 2/École Centrale de Lyon (Mar 2009)
7. Georgeon, O., Mille, A., Bellet, T., Mathern, B., Ritter, F.: Supporting activity modelling from activity traces. Expert Systems (Jun 2011)
8. Laflaquière, J., Settouti, L.S., Prié, Y., Mille, A.: A trace-based System Framework for Experience Management and Engineering. In: Second International Workshop on Experience Management and Engineering (EME 2006) in conjunction with KES2006 (Oct 2006)
9. Raptis, M., Kirovski, D., Hoppes, H.: Real-time classification of dance gestures from skeleton animation. In: Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation (August 2011)
10. Roth, E., Bennett, K., Woods, D.: Human interaction with an intelligent machine. Cognitive engineering in dynamic worlds 20 (1987)
11. Settouti, L.S., Prié, Y., Champin, P.A., Marty, J.C., Mille, A.: A Trace-Based Systems Framework : Models, Languages and Semantics. Research report, LIRIS, SYSCOM (2009), `http://hal.inria.fr/inria-00363260`
12. Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., Blake, A.: Real-time human pose recognition in parts from a single depth image. In: CVPR (2011)
13. Suma, E., Lange, B., Rizzo, A., Krum, D.M., Bolas, M.: FAAST: the flexible action and articulated skeleton toolkit. In: IEEE Virtual Reality. pp. 245–246. Singapore (Mar 2011)
14. Woods, D., Roth, E.: Aiding human performance ii: From cognitive analysis to support sytems. Le Travail Humain 51(2), 139–172 (1988)