In the Proceedings of the 6th International Conference on Virtual Systems and MultiMedia (VSMM'2000), Intelligent Environments Workshop (published by IOS Press, USA; ISBN 1-58603-108-2), pages 703-713, Gifu, Japan, October 2000.

Action Recognition: How Intelligent Virtual Environments Can Ease Human-Machine Interaction

Didier Verna
EPITA / LRDE, 14-16 rue Voltaire, 94276 Kremlin-Bicêtre cedex
mailto:didier@lrde.epita.fr
http://www.lrde.epita.fr

March 3, 2001

Abstract

This paper describes a research that has been conducted in the field of cognitive assistance to human-machine interaction in virtual environments. The idea is to design a system which, bearing in mind the actions performed by the operator at present and the current state of the environment, attempts to determine the global operation that the user is in the process of executing, and eventually takes control of the same process in order to complete it automatically. This idea implies the conception of an action recognition mechanism based on a specific knowledge representation model. This mechanism is implemented in a computer demonstrator, known as the `toast` system, which is also presented.

1 Introduction

Current virtual reality systems must establish a compromise between efficiency and usability: joysticks or trackballs, although they are easy to use, prevent the user from acting in a natural manner. On the other hand, exoskeletons and force-feedback arms can provide good immersive feelings, but these however are cumbersome devices. In order to find a solution to this sort of dilemma, two approaches are possible: the first, rather applicative, approach would be to improve the quality of human-machine interaction by making interfaces more user-friendly. The second, rather theoretical, approach would be to assist the person in his work by diminishing his workload and thereby reducing the quantity of interaction necessary. Our work deals with the idea of assistance and is therefore of relevance to the second approach mentioned above. We analyze the idea of assisting the human being within a virtual reality context, and we particularly show the benefit of such a concept of action recognition as a possible source of intelligent assistance.

Firstly, we give an overview of the functionalities we want to obtain, and we describe the context in which the system is supposed to be used. In particular, we clearly position our goal with respect to similar yet different problems such as planification.

Secondly, we present a knowledge representation model used to describe the actions an operator may undertake. We also establish a set of criteria that allow our action recognition engine to evaluate the probability of a particular action which it knows, to be the one actually undertaken by the operator.

Thanks to the level of genericity of our system, we demonstrate that whereas it was not originally designed in this end, the system can be extended to such features as action simulation and action execution with error tracking, all of this at a very low cost.

Finally, we describe a computer demonstrator known as the `toast` system, which implements the proposed action recognition mechanism, in addition to its simulation, execution and error tracking capabilities.

2 Why Doing Action Recognition?

Considering the state of the art in virtual reality systems (pure virtual reality systems [2, 12], augmented reality systems [1, 14], computer aided tele-operation systems [6]...), the main problem we are faced with is the problem of user interaction: a poor interaction can have critical effects on the operations, and can even cause physical pain [4]). As suggested in the introduction, a compromise must be established between the complexity of the interfaces and their usability. In tele-robotics for instance, two philosophies exist to help the operator interact with a robot: either virtual reality immersive devices are used, which leads to the field of tele-operation [5], or more intelligence is given to the robot, which leads to the field of autonomous robotics. This constitutes a kind of paradox because:

> the purpose of immersive systems is to give more *technical* abilities to the operator, for instance, being able to control very precisely a remote robot. This, however, doesn't belong to the natural capabilities of a human, but rather to a machine.
>
> the purpose of autonomous robotics is to give more *intelligence* to the robot, for instance, being able to cope with unexpected situations without the intervention of an operator. This, however, doesn't belong to the natural capabilities of a machine, but rather to a human being.

While investigating on the notion of assistance to human-machine interaction in virtual environments [13], it appeared to us that an intermediate approach would be worth trying: what could we do to help the operator without giving him extraordinary technical powers, and without giving the machine extraordinary reasoning skills ? A particular form of action recognition was then envisioned: consider for example a system in which an operator is controlling a robotic arm that is able to manipulate remote objects. Consider further that we *intentionally* don't want a tele-presence system (so we keep simple interaction devices only, for instance a joystick and a 2D visual feedback), and we *intentionally* don't want an autonomous system (so the robot must stay mostly under the control of the operator). The idea is then to design a system which is able to recognize relatively low-level, yet technically difficult actions (like object grasping), and takes the control of the execution only when there is no more complex behavioral decisions to make (like obstacle avoidance).

A typical scenario would then be the following: the operator starts moving the robotic arm in order to grab an object. A joystick is not very well suited to do this, but that does not matter, since imprecise motion is acceptable. When the robotic arm is close enough to the object, the assistance system recognizes the action and takes control of the remaining work. Its execution model is rather simple, but that does not matter since there are no more major difficulties left.

This particular form of action recognition is exactly the direction we took in our research (this particular example is actually implemented in a computer demonstrator known as the `toast` system, an acronym for "Tele-Operation Assistance System", see the last section), namely, trying to figure out to what extent useful assistance systems can be designed while keeping relatively simple models. This also explains why our research does *not* belong to planification. Planification takes place at a higher level, closer to autonomy [10, 3].

3 Knowledge Representation Model

3.1 Background

From a general point of view, the problem of knowledge representation is a very old one. It also has been addressed in different disciplines, such as cognitive science, artificial intelligence, robotics etc. Important work started with Norman [7] in cognitive science, followed by Sacerdoti [9] and Van Lhen & Brown [10] in planification.

Cognitively speaking, actions are traditionally represented as a set of three components: the action *prerequisites*, the action *progress*, and finally the action *result*. The action prerequisites is a set of initial conditions that are required in order for the action to start. The action result is the *declarative* part of the action: it expresses a state change consecutive to the execution of the action. On the contrary, the action progress is the *procedural* part of the action: it proposes a method to reach the goal.

There are two main reasons for splitting the concept of action in an “action result” and an “action progress”. Firstly, there are goals that can be reached by different methods: for instance, in order to heat the living room, one could either make a fire in the fireplace, or turn on the heater. Secondly, a single action can be executed in order to reach different goals: for instance, one could make a fire in order to heat the living room, but also just to burn down old newspapers.

Most formalisms proposed to represent actions, in particular Richard’s procedural networks [8], use oriented trees. That is also the base of our approach, as we will see in the sections below.

3.2 Generic Structure: Node Typing

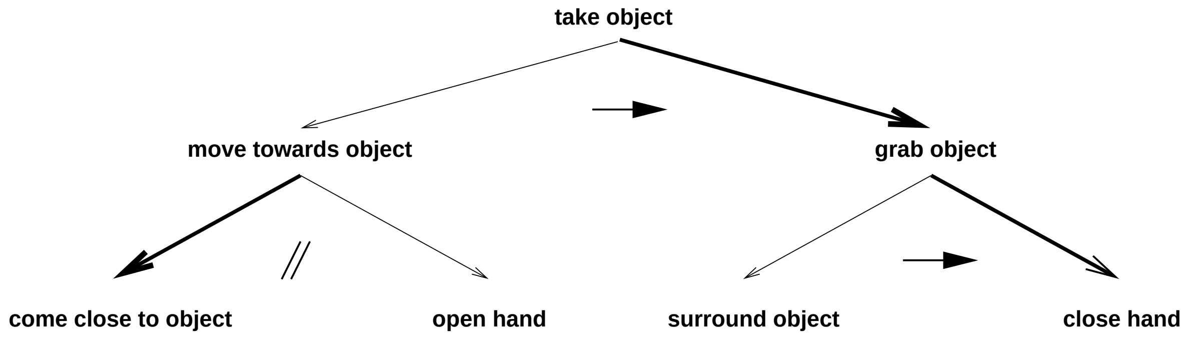
In order to build our model, a small experiment was conducted, consisting roughly to ask people to perform a certain operation (like taking an object), and then to verbalize what they had done. The verbalization process was reiterated until a satisfactory level of detail was reached. It appeared that people described the actions as a sequence of lower level actions. For instance, “take the object” was described as “Move the arm towards the object, and then grab it” etc.

In our knowledge representation model, actions are consequently described in the form of trees. Given one particular tree, the root node represents the particular “top-level” action that we are considering, and each sub-node in the tree represents a sub-action, that is, a “step” in the top-level action’s progress.

Please note that there are no semantic distinction between the top-level action and the sub-actions. Any action can be considered as a top-level one in a particular tree, and conversely can happen to be a sub-action in another tree. For instance, the action “take this object” can be considered as the top-level one, or as a sub-action in the action “transport this object somewhere else”.

We also need some kind of temporal ordering, in order to represent the fact that sub-actions are sometimes executed in a precise sequence. In the conducted experiment, people basically described (sub-)actions being performed one after each other, or in parallel. This

Figure 1: An action tree Example



suggests two kinds of actions, hence two types of nodes: actions whose development is made of sub-actions executed one after each other, and actions whose development is composed of sub-actions that can be executed simultaneously.

In the remaining of this paper, we will speak of an action **mode** to describe the way its sub-actions are traversed. Actions can be of a *sequential* or *parallel* mode. In order to illustrate this structure, we provide on figure 1 a graphical representation of the action "take an object", as most subjects of our experiment described it. In sequential actions, a "→" appears between each sub-actions while in parallel ones, the sign "//" is used. The rest of the figure will be explained later.

The action on figure 1 is primarily composed of two sub-actions: the parallel action "move the arm towards the object" and the sequential action "grab the object". An additional sub-level of description is also provided.

3.3 Extended Structure: Branch Typing

The proposed model is satisfactory to represent people's descriptions of actions, yet it is not sufficient to be usable in an action recognition engine. There are mainly three deficiencies in this representation:

It is possible, at the time of beginning an action, that some of its sub-actions don't need to be executed because their goal is already achieved. An action recognition engine using our model in its current state would blindly try to detect all sub-actions in all cases.

A human being is naturally capable of "error recovery", that is, resuming the execution of an action from a previous step, if an incident happens. An action recognition engine using our model in its current state wouldn't have a way to figure out why and when something needs to be redone.

Conversely, a human being knows when some steps in an action are *temporary* goals that need to be reached only in order to proceed, but are not part of the final result. For instance, in an object transportation action, the fact that the object is grasped at some point is only an intermediate result than can (must) be forgotten in the end.

These points demonstrate that we need more information on each action. More precisely, we need to know exactly under which circumstances a sub-action's goal needs to be preserved in order to proceed. Here again, the conducted experiment demonstrated that the sub-actions goals were used in only two different ways: some sub-goals need to be preserved permanently (they are actually part of the final goal), while some others need only be preserved until the parent action terminates.

In the remaining of this paper, we will speak of a branch **quality** to describe the way sub-goals are handled in a particular action tree. In such a tree, branches can be of a *persistent* or *auxiliary* quality. A persistent branch is a branch whose goal needs to be preserved permanently, while an auxiliary branch is a branch whose goal needs to be preserved for the duration of the upper-level action only.

Please note that we are typing the *branches* (not the nodes or the actions) of a tree. That is because theoretically, it is perfectly possible for a single action's goal to be used in a persistent way in a higher level action, and in an auxiliary fashion in another. In order to illustrate this extension of our model, we provided on figure 1 some additional graphical clues: persistent branches are represented in bold, while auxiliary branches appear in thin.

You can see for instance that the branch containing the "move towards object" sub-action is auxiliary because once the next one, "grab object" is accomplished, it can be forgotten. On the contrary, the branch containing the "grab object" action is persistent, because if it was to be invalidated at some point, the upper one would also be invalidated.

3.4 Formalization

Our knowledge representation model has a very precise declarative semantics. It would be too long to give it here, but it is described at length by Verna [11]. Please note however that this model is very simple yet powerful, as we will see in the next section. Actions representation obey a strict syntax which can be described thanks to a four line BNF grammar as shown below:

```
A ::= S | P | t
S :: = B [-> B] ... [-> B]
P :: = P [// P] ... [// P]
B :: = Pers(A) | Aux(A)
```

An action (`A`) can be sequential (`S`), parallel (`P`) or terminal (`t`). `B` denotes a branch, which can be either persistent (`Pers`) or auxiliary (`Aux`). Terminal actions are those which end the development of an action tree. They usually depend on the system being used. For instance, in the `toast` system, the operator controls a virtual robotic arm thanks to a simple 2D joystick. The system provides five terminal actions: move the arm vertically, rotate the arm around the vertical axe, open and close the plier and rotate the fist (1D). All other actions are built on top of these.

3.5 Specificities

Compared to other approaches of similar problems, our model features several specificities that are worth mentioning:

Mutual Reference. The two concepts grounding our model are the concepts of action (or node) and branch, and each one is actually defined in terms of the other. This makes our model completely generic: only terminal actions depend on the system being used and need to be hardwired. For instance, in `toast`, all non-terminal knowledge is written in a Lisp dialect, interpreted at *runtime*, and thus can be modified without requiring any recompilation.

Actions Reusability. Actions, once declared, can be used (as sub-actions) in different trees, or even several times in the same tree. The action "open the hand" is a typical example of this, as it appears twice in the top-level action "transport an object somewhere else" (once in the "grab the object" sub-action described formerly, a second time to release the object. This contrasts with most other existing cognitive models.

Implicit Goals Representation. While traditional approaches make a distinction between the action progress and the action goal, our model doesn't. This is both because we *intentionally* represent actions in the simplest possible method, and because of the structure of our model: an action's goal is *implicitely* described as a composition of its sub-actions'goals, which, in the end, turns into a composition of terminal actions'goals (the only ones to be hardwired).

4 Action Recognition

4.1 Likeliness Rate

As previously mentioned, our action recognition engine's main job is to perform a matching between the actions it knows, and the ones currently undertaken by the user: we need to evaluate "how close" the known actions are to what is currently being executed by the user. In this end, we define a measure, that we call the "likeliness rate". The computation of this rate for each known action permits to decide whether an action should be recognized, and which one in that case.

Given the structure of our knowledge representation model, and in particular, given the fact that all actions are decomposed in trees of sub-actions, it is natural to define the likeliness rate of an action in the same way: for a given action, its likeliness rate must be defined in terms of a composition of the likeliness rates of its sub-actions. Therefore, we have to cases to envision: sequential actions, and parallel actions.

Parallel Actions. In a parallel action, all sub-actions are independent of each other and may be executed simultaneously. This suggests an additive composition of the likeliness rates: the likeliness rate of a parallel action should be a function of the sum of the likeliness rates of its sub-actions. Intuitively, this means that a parallel action is likely in the condition that all its sub-action also be likely.

Sequential Actions. The case of sequential actions is a little bit more tricky. In a sequential action's progress, sub-actions are supposed to be performed one after each other, the beginning of a sub-action being conditioned by the termination of the preceding one. This suggests a multiplicative composition of the likeliness rates: the likeliness rate of a sequential action would be a function of the product of the likeliness rates of its sub-actions. However, this is not completely true: it more correct to say that while a sub-action is being executed, not only the next one, but actually the *whole remaining sequence* cannot start. Therefore, a more correct definition for the likeliness rate should be a function of the product of the likeliness rates of the *sequences* of sub-actions.

Action Weighting. Before giving a complete definition of the likeliness rate, a final adjustment needs to be done. Consider the representation of the action "take an object" given in figure 2. It is actually semantically equivalent (this can be proven) to have the last terminal action `Close(OBJECT)` directly linked to the top-level one. In that case, the three sub-actions of the top-level action would clearly be of quite different complexities. Therefore, we should not consider their likeliness rates at the same level in the computation of the top-level rate. We thus need to weight the nodes of the tree in order to take each action's complexity into account. The weighting we use is actually very simple: a node's weight is the sum of its immediate sub-nodes'weights. The complexity of the actions hence increases while we are moving towards the top-level one.

We now give the precise definition of the likeliness rate of an action. This is exactly the computation that is being used in the `toast` system. Let A be an action developed in $\{A_1, A_2 \ldots A_N\}$ sub-actions. Let ω_A be the weight of A, that is, the sum of the weights ω_i of all sub-actions of A. Let furthermore $\tau_i \in [0,1]$ be the likeliness rate of the sub-action A_i. When A is a parallel action, its likeliness rate is then given by:

$$\tau(A) = \frac{\sum_{i=1}^{N} \omega_i \tau_i}{\omega_A} = \frac{\sum_{i=1}^{N} \omega_i \tau_i}{\sum_{i=1}^{N} \omega_i}$$

Consider now a sequential action. The likeliness rate of the *sequence* $\{A_i, A_{i+1} \ldots A_N\}$ of sub-actions is given by:

$$\lambda_i = \tau_i.(\omega_i + \lambda_{i+1}) = \tau_i.(\omega_i + \tau_{i+1}.(\omega_{i+1} + \ldots))$$

And finally, the likeliness rate of A is given by:

$$\tau(A) = \frac{\lambda_1}{\omega_A} = \frac{\omega_1 \tau_1 + \tau_1.[\omega_2 \tau_2 + \tau_2.(\ldots + \omega_N \tau_N)]}{\sum_{i=1}^{N} \omega_i}$$

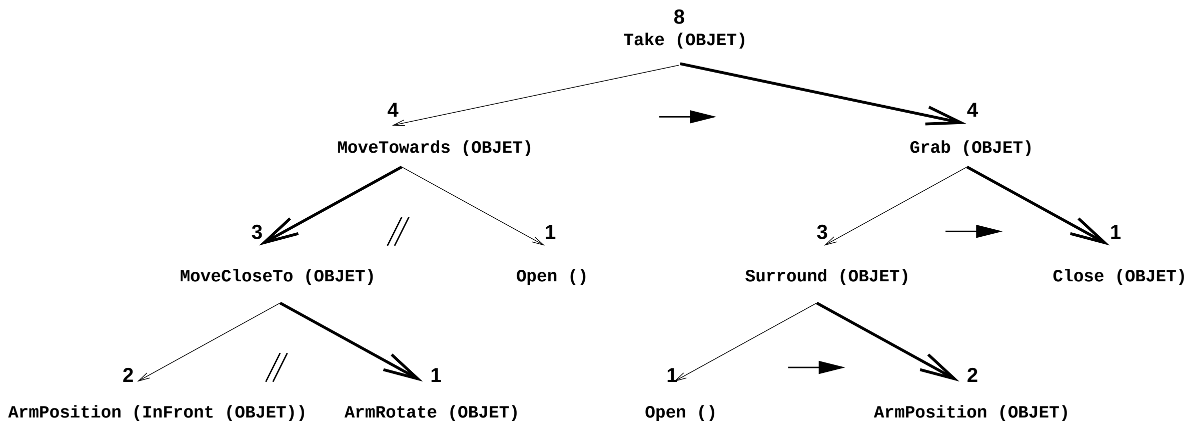
4.2 Terminal Actions

As we said before, terminal actions are the only hardwired actions in a given system. Their likeliness rate computation is consequently also hardwired, and is quite different from the definitions above. We propose to demonstrate how such rates can be elaborated through an example action implemented in the `toast` system: the `ArmPosition()` terminal action. This action (whose execution can be controlled through joystick motion) allows the user to move the virtual robotic arm in the vertical plan, to a specified 2D position, for instance, the position of an object in the scene.

Activation Rate Suppose now that the user is moving the arm towards an object present in the scene. In a first approximation, we are tempted to say that the closer to the object we are getting, the more likely the `ArmPosition(object)` action is to be undertaken. However, that is not correct: the arm could for instance come very close to the object (like passing above), without actually moving *towards* it. In such a case, it would be wrong to recognize this action.

This example demonstrates that for terminal actions, it is important to take into account the proximity of the goal, but also the proximity of the *method*: a terminal action is “likely” not only if the goal is not too far away from being reached, but also if the hardwired execution method for this action is close to what the user is currently doing. What are these “hardwired” execution methods then ? Since all such methods are for terminal actions only, and since these actions are relatively low-level, the answer is easy: the simplest ones. For instance, the `ArmPosition(object)` hardwired execution method will just consist in giving the arm a speed vector making it go to the required position in a straight line.

Figure 2: A `toast` Action Example



Likeliness Rate Continuing the example action above, the likeliness rate of this action takes into account both the distance to the object (this is the proximity of the goal) *and* the angular distance between the current speed vector of the arm and the vector pointing at the object (this is the proximity of the method).

Weighting. Finally, a last point must be resolved: we need to define the weights of the terminal actions. Since weights are actually a measure of complexity of the actions, we chose to define the terminal actions' weights as the "dimension" of these actions. For instance, in the `toast` system, the action `ArmPosition()` operates in a plan, so it has a weight of 2. On the contrary, the action `ArmRotate()` operates in one dimension only, so it has a weight of 1.

As an example, the action "take an object" is given again but as it is exactly represented in the `toast` system, with terminal actions and weighting represented.

4.3 Corollaries

Describing the algorithms used to implement our action recognition engine would be going in too much detail here. Again, these algorithms are described at length by Verna [11]. Let us however mention that once this knowledge representation model was settled, and the action recognition engine implemented, we found out that it was very easy to extend the system to such features as action *execution*, action *simulation* and even *error tracking and recovery*.

Execution. The algorithms used to recognize actions already take into account all the characteristics of our knowledge representation model: the actions'modes are used in order to determine which actions should (not) be subject to a recognition process at a particular time, and the branches'qualities are used to decide at what time exactly a particular action recognition process should start. The interesting consequence is that these algorithms work exactly as execution algorithms, apart from the fact that state changes occur on likely / not likely transitions instead of achieved / not achieved ones. In other words, the real execution algorithms can be obtained by actually *simplifying* the action recognition ones: one has just to turn the likeliness rates into boolean variables indicating whether an action's goal is reached or not.

Simulation. In a similar vein, once execution algorithms are obtained, it is very simple to turn them into simulation ones: instead of actually executing the actions, one just simulates their behavior on virtual copies of the objects for example. But how is it possible to *foresee* the result of such or such action ? That comes from the fact that only terminal actions'results need to be pre-computed. And in turn, these terminal actions happen to know their execution method already because we needed them for recognition purpose. For instance, consider again the `ArmPosition()` terminal action in the `toast` system. We saw that in order to compute its likeliness rate, we needed to compute the speed vector leading directly to the object first. It is therefore trivial to use this *already performed* computation to execute the action, or just to simulate it.

Error Tracking / Recovery. Finally, with this very same computation in hand, we are also able to "predict" the state of the action at the next time step. For instance, knowing the speed vector of the arm, we can compute its position at $t + \delta t$. During an action execution, we can then compare the expected result with the one actually obtained at each time step. If a discrepancy is observed, then the action probably went wrong and should be started again.

All of this demonstrates that the genericity of our model permits to extend very easily the action recognition engine to features that were not initially expected. In the `toast` system, the additional cost of these features is evaluated to less than 10% of the total amount of code.

5 A Sample Scenario

Before concluding this paper, we would like to present a typical scenario that has been implemented in the `toast` system (see figure 3), and which demonstrates the main features of our action recognition engine (apart from the error tracking / recovery facility).

1. In addition to the robotic arm controlled by the operator, two objects are initially present in the scene: a glass and a symbolic square sticked to the floor. `toast` is not aware that the symbolic square cannot be grabbed, but that is on purpose... The operator has only a mouse to control the virtual robotic arm: a joystick is simulated on the screen.

2. The operators wonders if the situation is simple enough for the system to succeed in automatically grabbing the glass. In order to get the answer to this question, (s)he request a simulation of this action. Since `toast` doesn't know that only the glass can be grabbed, it asks the operator to select the object (s)he is interested in. The two objects become preselected on the screen.

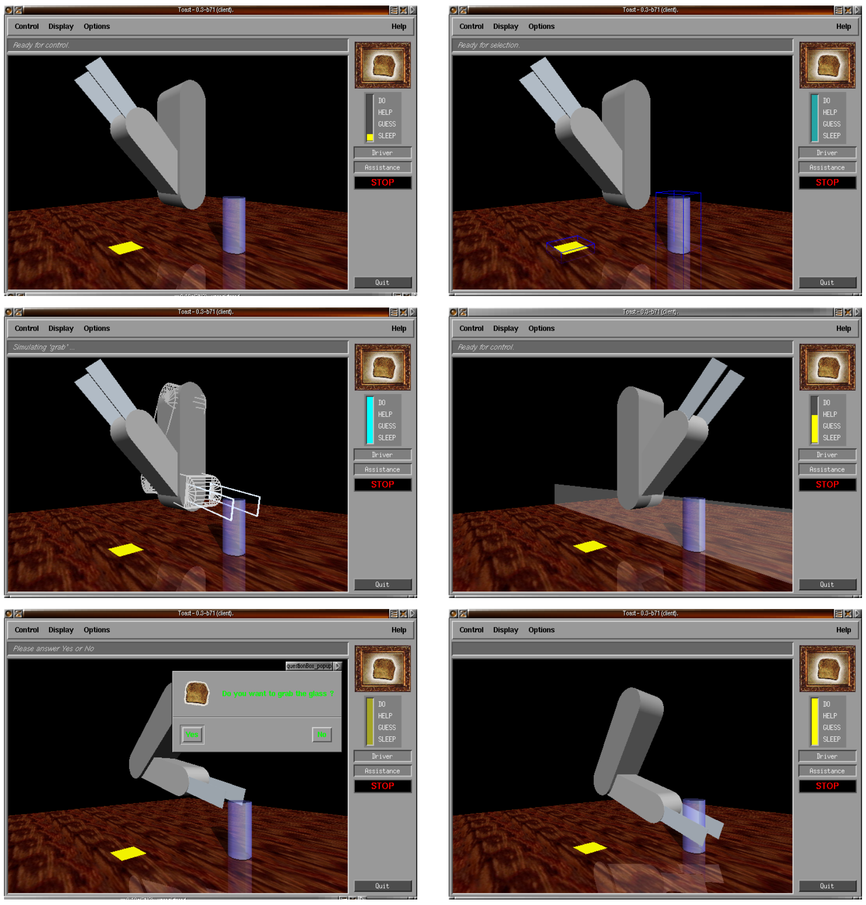
3. The simulation then goes on (the arm is drawn in wireframe), and the operator discovers that the automatic execution would fail, because the wireframe arm visibly intersects with the glass. As a consequence, the operator decides to start the movement manually.

4. The first thing to do is to rotate the arm in order to reach a proper alignment with the glass. During this operation, which normally would be very imprecise, provided with only a 2D visual feedback and a mouse for control device, `toast` displays a virtual translucent plan corresponding to the arm's vertical plan. This makes it much easier to reach a satisfactory alignment.

5. The operator then starts to come closer to the glass. At a certain point, `toast` decides that the action of grabbing the glass is very likely, and asks for a confirmation.

6. The operator answers "yes", and finally, `toast` performs the end of the action automatically.

Figure 3: A Sample Scenario in `toast`



6 Conclusion

In this paper, we have presented a research in the field of intelligent assistance to human-machine interaction in virtual environments. This research was oriented towards a special form of action recognition whose purpose was to investigate intermediate and interactive solutions between fully slaved and fully autonomous environments.

The proposed knowledge representation model is very simple (its formal description takes only a few lines) and completely generic: only the terminal actions may vary from system to system. It also has no particular knowledge of the scene, apart from what is strictly relevant in the current context. That makes the system fast, not resource consuming and very reactive. Moreover, knowledge acquisition is dynamic: non terminal actions are described in a Lisp dialect for which an interpreter is embedded in the action recognition engine (written in C and C++).

Thanks to the level of genericity of our system, we also demonstrated that whereas it was not originally designed in this end, the system can be extended to such features as action simulation and action execution with error tracking, all of this at a very low cost. One can then reach a high level of interactivity, even with very simple peripherals such as joysticks, provided that interaction is conducted through an intelligent virtual environment. Many industrial applications can benefit from these ideas, notably in the fields of assistance to handicapped persons and computer-aided tele-operation systems.

References

[1] Ronald Azuma. A survey of augmented reality. In *Presence*, volume 6/4, pages 335–385. MIT Press, August 1997.

[2] Grigore Burdea. virtual reality systems and applications. In *Electro'93 International Conference*, page 164. Edison, April 1993.

[3] G. Giralt, R. Chatila, and R. Alami. Remote intervention, robot autonomy and teleprogramming: Generic concepts and real world application cases. In *International Conference on Intelligent Robots and Systems*, pages 314–320. IEEE/RSJ, IEEE Computer Society Press, 1993.

[4] Rob Kenedy, Susuan Lanham, Julie Drexier, Catherine Massey, and Michael Liliental. A comparison of cybersicknesses, incidences, symptom profiles, measurement techniques and suggestions for future research. In *Presence*, volume 6/6, pages 639–644. MIT Press, 1997.

[5] A. Kheddar, C. Tzafestas, P. Blazevic, and P. Coiffet. Fitting tele-operation and virtual reality technologies towards teleworking. In *FIR'98, 4th French-Israeli Symposium on Robotics*, pages 147–152, Besaçon, France, May 1998.

[6] W. S. Kim. virtual reality calibration and preview / predictive displays for telerobotics. In *Presence*, volume 5/2, pages 173–190. MIT Press, 1996.

[7] D.A Norman and D.E. Rumelhart. *Explorations in Cognition*. Freeman and Co, San Francisco, 1975.

[8] Jean-François Richard. *Les activités mentales*. Armand-Collin, 1990.

[9] E.D Sacerdoti. *A Structure for Plans and Behavior*. Elsevier Computer Science Library, 1977.

[10] K. Van Lhen and J.S. Brown. Planning nets: a representation for formalizing analogies and semantic models of procedural skills. *Aptitude Learning and Instruction*, 1980.

[11] Didier Verna. *Télé-Opération et Réalité Virtuelle: Assistance à l'Opérateur par Modélisation Cognitive de ses Intentions*. PhD thesis, ENST, 46 rue Barrault, 75013 Paris, France, February 2000.

[12] Didier Verna and Alain Grumbach. Can we define virtual reality? the ***MRIC*** model. In Jean-Claude Heudin, editor, *Virtual Worlds 98*, Lecture Notes in Artificial Intelligence, pages 29–41. Springer-Verlag, 1998.

[13] Didier Verna and Alain Grumbach. Sémantique et localisation de l'assistance en réalité virtuelle. In *GTRV'98*, pages 105–112, 1998.

[14] Didier Verna and Alain Grumbach. Augmented reality, the other way around. In M. Gervautz, A. Hildebrand, and D. Schmalstieg, editors, *Virtual Environments'99*, pages 147–156. Springer, 1999.