Defining and Executing Practice Sessions in a Robotics Virtual Laboratory

Lourdes Muñoz-Gómez, Moisés Alencastre-Miranda, Isaac Rudomín
Instituto Tecnológico y de Estudios Superiores de Monterrey. Campus Estado de México.
Carretera Lago de Guadalupe Km 3.5, Atizapán de Zaragoza, Estado de México, CP 52926
{00709911,00471699}@academ01.cem.itesm.mx, rudomin@itesm.mx

Abstract

There is a lack of practice sessions in distance learning courses, in which students can use the theoretical concepts seen in class. In this paper we present an implementation of a Robotics Virtual Laboratory in a Networked Virtual Environment (net-VE) with the capability of defining practice sessions that can be executed in a visual simulation of the environment. Several manipulator robots, mobile robots and furniture can populate the environment. A programming language for manipulators and mobile robots is presented. The language contains instructions to provide coordination between robot programs; an interface for defining a task as a set of conditions for executing programs has also been implemented. Teachers can assist students either by collaborating during the execution of the practice or writing a rule-base script for observing movements of the robots.

1. Introduction

Distance learning has become a good solution for people who need a flexible teaching system without spatial or temporal limitations. Information technologies provide tools that allow communication between teachers and students. Although by using these tools good courses can be implemented, there are usually no practical sessions complementing information given in the course [1]. On the other hand the investment needed to install a real laboratory is high and not all institutions can provide equipment for the student's use: this is the case in robotics laboratories, automatic control laboratories and so forth. Even when such laboratories are available, the investment has to be protected, so the laboratory must have a qualified assistant keeping guard, insuring the correct use of the equipment; students have to work carefully in order to avoid disasters causing irreversible damage to laboratory and equipment. The idea of having a Virtual Laboratory is to provide training and learning tools for students, improving distance learning programs, reducing investment and providing a flexible experimental framework in which there is a diminished risk of damage caused by accidents.

For our purpose, a Virtual Laboratory is a heterogeneous and distributed environment for accessing virtual or real equipment from remote places. The advantages of a virtual laboratory include: cost reduction, transparency in simulation, expandability and risk reduction among others [2]. When using virtual equipment, the virtual laboratory consists of a Networked Virtual Environment (net-VE) allowing users to interact with each other.

A net-VE is a software system where multiple users interact in real time, wherever these users may be, including geographically different places [3]. Typically, each user has access to a computer that provides a graphical interface for the virtual world. A net-VE has a shared sense of space, a shared sense of presence, a shared sense of time, a communication mechanism and a sharing mechanism [3].

In the work described in this article, we use an object oriented and distributed architecture in order to have a net-VE, allowing teachers and students interact and collaborate in a visual simulated world. The system includes several manipulators and mobile robot models that can be selected by users to create worlds in which practical exercises can be accomplished. The teacher is able to define a practice session, and students are able to execute it and save the information for future consultation.

A programming language is defined that can be used to program both manipulators and mobile robots. We have also implemented an interface for teaching positions to manipulators (as is done with real robots with the teach pendant). Once defined, these positions can be parameters for the "move" instruction.

The remainder of this paper is organized as follows: in Section 2 work related to net-VE and virtual laboratories is discussed; in Section 3 the net-VE used for the robotics virtual laboratory is briefly described, Sections 4 and 5 explain how to define and execute practice sessions in our system; in Section 6 examples of practice sessions are shown and finally in Section 7 and 8 conclusions and future work are discussed.

2. Related Work

Originally net-VE architectures were developed for military purposes (combat training simulations); examples of these architectures are DIS [3] and HLA [3]. Subsequently, other net-VEs architectures where developed with academic purposes, allowing users interaction in a shared room in which can take place virtual meetings or conferences. Two examples of this kind of architectures are DIVE [4] and INVITE [5]. In this work we use a modified version of OODVR, another net-VE architecture. In [6] a more detailed comparison between OODVR and other net-VE systems is presented.

Several different applications have been called "virtual laboratories" in different areas (chemistry, physics, power electronics, control systems, robotics, manufacturing, etc). Some of these virtual laboratories can't be accessed by more than one user at the same time [1, 7]; these are stand- alone applications do not permit collaborative work. Other types of virtual laboratories have been created having as their main purpose to be used by a lot of people around the world, so these labs have been implemented on the Internet so as to permit wide access to them. Examples of this type of labs can be found in [8, 9, 10, 11, 12]. However, even if these virtual labs are on the Internet and multiple users can access the applications, there is no collaboration between all the users that are working at the same time: every user is working by himself without knowing anything about the existence or activities of the others. Finally some virtual laboratories have been implemented using net-VEs, and these laboratories provide collaborative environments in which users can interact. In the case of E-manufacturing [13], several users can plan manufacturing process and define layouts. Another example is a testbed for mobile robots [14]. However these last two examples do not allow the professor or student the possibility of defining or executing practice sessions.

3. Object Oriented Distributed Virtual Reality Architecture applied in Virtual Laboratories (OODVR++a)

The Object Oriented Distributed Virtual Reality Architecture (OODVR) is a net-VE architecture developed for general-purpose applications. This architecture was modified in order to provide a collaborative visual simulated environment for robotics virtual laboratories and this last one was called OODVR++a.

OODVR was designed and implemented having as an objective a distributed environment for simulations, where the complete simulation is integrated by different elements. This architecture assumes the participation of several computers in a complete simulation, each called "participant", that can contribute with one or more "entities" or modules to the simulation. The entities belonging to a given participant are called "local entities" if executing locally and "proxy entities" if executing on the other machines [6].

OODVR was modified in order to have more tools that would allow using it in the implementation of robotic virtual laboratories. OODVR++a allows the entrance of entities at any time during the simulation. Several modules were also integrated to OODVR to create OODVR++a:

- Database.
- Chat.
- Simple control mechanism.
- Simulation clock.
- Rule-base scripts.

The database was integrated in order to save robot configurations, environments, programs and practice sessions. This saved information allows consulting of previous practice sessions, loading unfinished programs and performing an asynchronous learning. In the case of robot configurations, the database has the information about degrees of freedom, 3D models, number of links, pieces of each link and type of each robot. This approach allows us to add different manipulators or mobile robots models to the database that can be used in the simulation without recompiling the complete application. At this time, in the database exist five manipulators (Mitsubishi EXR, CRSA465, Amatrol Jupiter XL, Puma 560, Amatrol ASRS) and two mobile robots (AmigoBot, Pioneer).

In order to provide a communication channel between users during simulation, a text chat was added and the users may collaborate in the practice sessions.

A simple control mechanism was implemented in order to allow only one user to control the same robot at the same time as occurs in real laboratories.

A simulation clock was also implemented to allow movement synchronization of local entity robots and proxy robots. This has to be done, because of latency on computer networks.

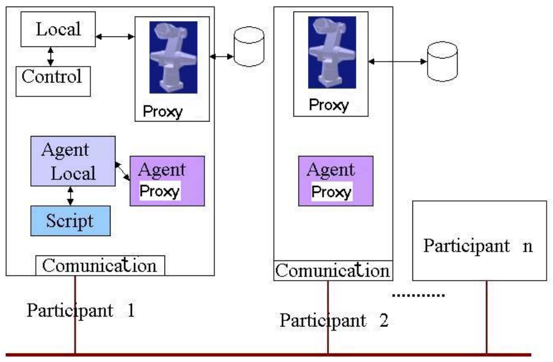


Figure 1. OODVR++a architecture.

Finally, an important feature of OODVR++a is the capability of inserting rule-base scripts in which users can implement conditions that will be checked every five seconds. These allow users to write simple agents for monitoring certain positions, velocities or movement in robots. The scripts only have to be placed in a directory and there is no need of compiling the application. Figure 1, shows the OODVR++a architecture.

4. Defining Practice sessions

In order to define a practice session, users must first create and save an environment in which the practice will be take place. The user or users simply add entities in the desired positions, after that only one user saves the environment as a new world that will be used later. Saving the environment avoids the creation of a new environment every time the simulation is run. Figure 2 shows an environment with several manipulators and mobile robots.

After the user has created and defined an environment, a new practice can be created; a dialog window asks for the user to specify:

- The name of the practice,
- The environment that will be used and
- The file containing the instructions of the practice.

5. Executing Practice sessions

Once the practice has been defined the users can begin the execution to complete the assignment described in the practice session. The execution can be done immediately after the definition or can be done at a different time if users in the simulation load previously defined practice sessions.

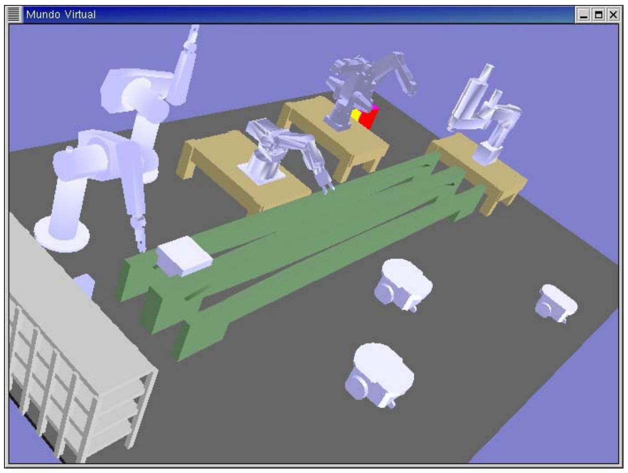


Figure 2. A world populated with manipulators and mobile robots.

Each user connected to the simulation can start working on a different robot; every one can program a robot just as a team working in the same physical laboratory would. When all the programs have been finished, only one user can define a task combining the programs; all users can see the execution of the task. The individual programs of each robot and the definition of the task can be saved in order to be loaded in a different simulation.

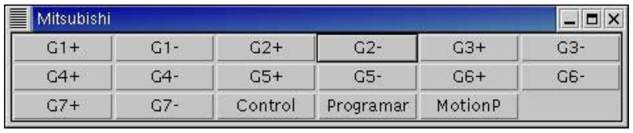
We next explain how to program the robots, how to define a task and how a script can provide a kind of assistance.

5.1. Programming Virtual Robots

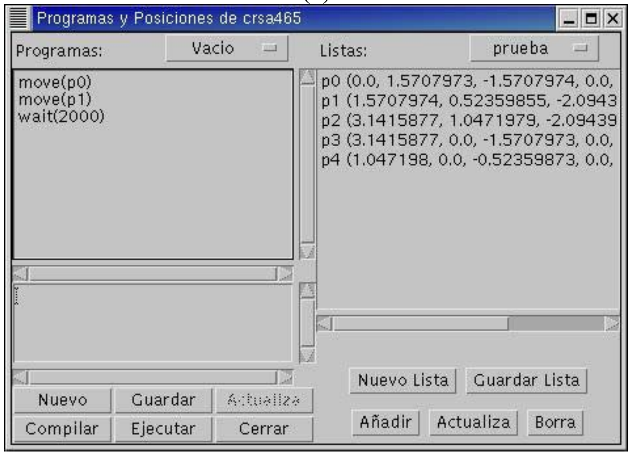
Each virtual robot (manipulator or mobile) in the visual simulation has an interface allowing users to teach positions (in the case of manipulators), to write a program, to compile and to execute that program on the virtual robot.

5.1.1. Teaching positions to a Robot. When people work with real manipulators, they use a teach pendant in order to define fixed positions that will be useful for programming a sequence of movements going to individual fixed positions. This particular feature is implemented in the virtual laboratory using a button window (one for each robot) to move each degree of freedom until the desired position is reached. After that another button has to be pressed indicating that this particular position has to be added in the current list of positions. Each robot can have several lists of positions associated to it; these lists can be loaded any time and can be used in all the practice sessions. The information related to the lists of position is saved on the database. In

Figure 3a we see the button window for a robot and in Figure 3b the window having a list of positions.



(a)



```
move(p0)
move(p1)
wait(2000)
```

```
p0 (0.0, 1.5707973, -1.5707974, 0.0,
p1 (1.5707974, 0.52359855, -2.0943
p2 (3.1415877, 1.0471979, -2.09439
p3 (3.1415877, 0.0, -1.5707973, 0.0,
p4 (1.047198, 0.0, -0.52359873, 0.0,
```

(b)

Figure 3. (a) Button window to control each degree of freedom.
(b) Programming window with a list of positions on the right side.

During the execution process each robot reaches a fixed position indicated in a list by interpolating the current position with the position specified position. The difference between the angles of each degree of freedom of the current position and the angles of each degree of freedom of the new position is calculated in order to know how much each link will move.

5.1.2. Programming Language. We decided to use the same programming language for both manipulators and mobile robots. The main reason is didactic; users have to learn only one language syntax. Using real robots, a different language for each one is needed. Depending on the robot system, the language used to program robots can be like an assembly language or a high level language. The main disadvantage of using a single language is having the need to translate the program for virtual robots from our language to each robot language. However an automatic translation of programs could be implemented to particular languages of real robots.

Users at any time during the simulation can create new programs and change existing programs; they also can compile and execute different programs on each robot. One of the main advantages of the language design is the fact that each single program is saved in a file and compiled "on the fly" when the user executes it the first time, so the complete simulation does not have to be compiled with a predefined group of robot programs.

Control structure Or instruction	**Description**
if ..... else	Decision control structure.
while	Control structure for cycling.
move (p)	Moves robot to position p.
move (f, g)	Moves degree of freedom *f, g* units
speed(v)	Defines speed *v* for the robot movements.
flag(v)	Assigns *v* to the robot flag.
wcelflag(n, v)	Writes *v* on the *n* th cell flag.
rcelflag(n, v)	Reads *n* th cell flag and keep the value in *v*.
wait(m)	Waits *m* milliseconds to execute the next instruction.

Table 1. Control structures and instructions.

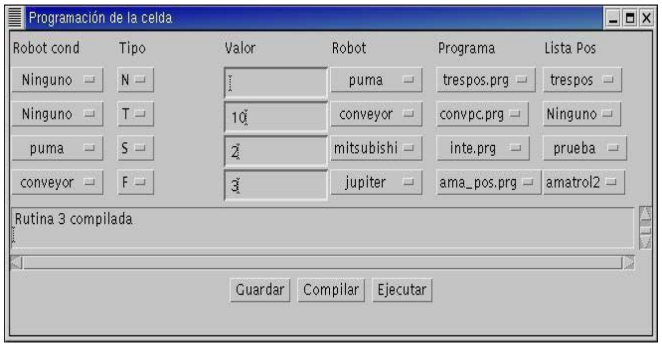
Each robot has a flag that can be set by the robot, but all robots in the world can read this flag, so the flag can be used to indicate different internal state of the robot. There also exists a ten–flag buffer for the environment (called the cell flags). This set of flags can be set and read by all the robots and helps the user in implementing coordination between programs. For example, a robot could set a value of '6' in the first position of the buffer, and other robots wait for that number '6' to begin a list of instructions. Table 1 shows a list of the control structures and instructions defined in the language. The language also permits declaring integer variables and arithmetic operations.

Each program is saved in a file and an entry on the database is also added to relate a program to a robot. All programs can be seen for any practice session.

Figure 3b shows on the left side of the window the program editor interface in the system.

5.2. Coordinating Robots: Defining a task

In the previous section we briefly described how the *rcelflag* and the *wcelflag* instructions can be used to coordinate program executions between robots. However we implemented another interface for defining starting conditions for a set of programs that will define a task, this is specially useful for defining time conditions and parallel execution of programs. Once users have finished with the individual programs for each robot, one user can define the task.



Robot cond	Tipo	Valor	Robot	Programa	Lista Pos
Ninguno	N		puma	trespos.prg	trespos
Ninguno	T	10	conveyor	convpc.prg	Ninguno
puma	S	2	mitsubishi	inte.prg	prueba
conveyor	F	3	jupiter	ama_pos.prg	amatrol2

Rutina 3 compilada

Figure 4. Interface for defining tasks. Four starting conditions, from top: N, T, S, F.

Figure 4 shows the interface in which appears as a table with the information that has to be filled. The first three columns define the starting condition, and the last three columns define the robot that will execute the program with a list of positions. There are four different starting conditions, note that if no condition is specified for all robots, all programs will be executed at the same time in parallel. The conditions are:

- None: Denoted with N, and this means there is no condition to begin the execution of the program. The program will start at the beginning of the task.
- Time: Denoted with T, and the user must specify a time t in seconds. This means that a program will be started after t seconds from the initial time in which the task began.
- Status: Denoted with S, and the user must specify another robot's state in order to begin the execution of the current program. We consider two different status: 1, a robot has started the execution of a program and 2, a robot has finished the execution of a program.
- Flag: Denoted with F, and the user must specify the flag of which robot has to be read with a specific value in order to begin the execution of the current program.

Users can save the defined task for a practice, for future consultations. No more than one task can be saved for a practice.

5.3. Assistance during practice session execution

During the execution of the practice, several users can be working at the same time, and the teacher can be running the same simulation in order to provide assistance to the students. The teacher can be a spectator only watching what students do, and using the chat interface, the teacher can communicate to the students providing them help or corrections.

However it is possible that teachers do not have enough time for attending to each of the practice sessions, so the system provides the possibility of adding a rule-based script in order to monitor certain actions of the students. The script is a JESS file (JESS [15] is a expert system shell in Java); teachers have access to the robot class methods in order to define rules. The use of these scripts for the moment is only limited to check some properties of each robot. If teacher needs a more intelligent assistant, other methods have to be implemented in the robot class, and more classes have to be developed to control or check other entities than robots.

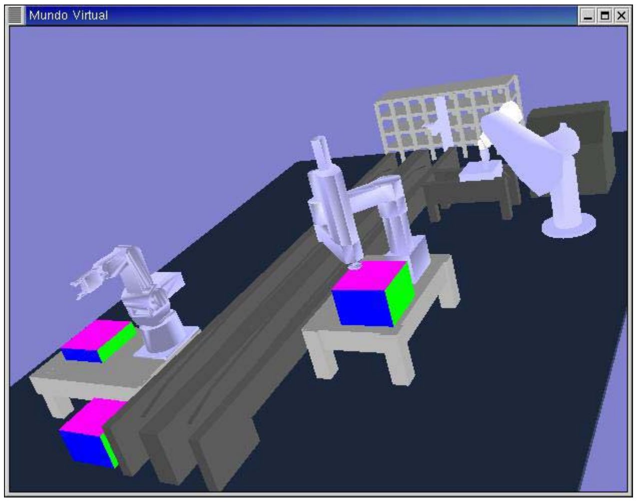
6. Practice Session Example

In this example, there is a complete manufacturing cell with three robots: a Mitsubishi EXR, an Amatrol JupiterXL and a Puma 560. Also in the environment there exist a conveyor band, two tables and a lathe. The user has to complete the following task:

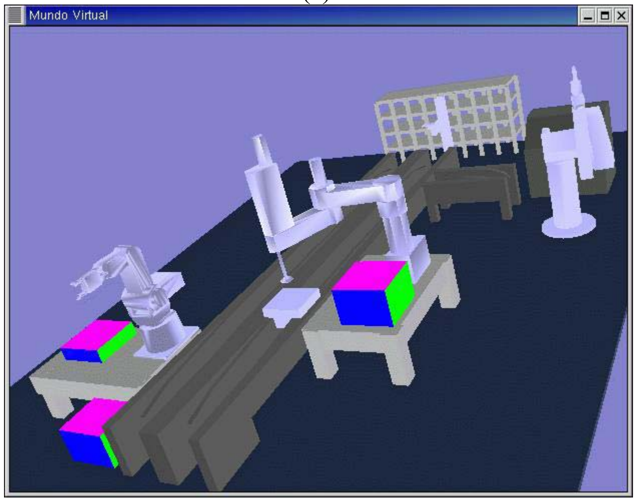
- The conveyor band must go to the Puma and wait.
- Puma must take a cylinder and put it inside the lathe and wait until a piece is finished.
- Puma must put the piece on the conveyor band.
- The conveyor band must go to the Jupiter Amatrol and wait.
- Jupiter Amatrol must take a stamp inside the box on its table, and put on the piece.
- The conveyor band must go to the Mitsubishi and wait.
- Mitsubishi must take a piece of cloth from the box next to it, go to the piece on the conveyor and clean it. Then Mitsubishi must throw the piece of cloth to the box on the floor.
- Finally the conveyor band must return to its original position.

A code fragment, for the conveyor band is shown:

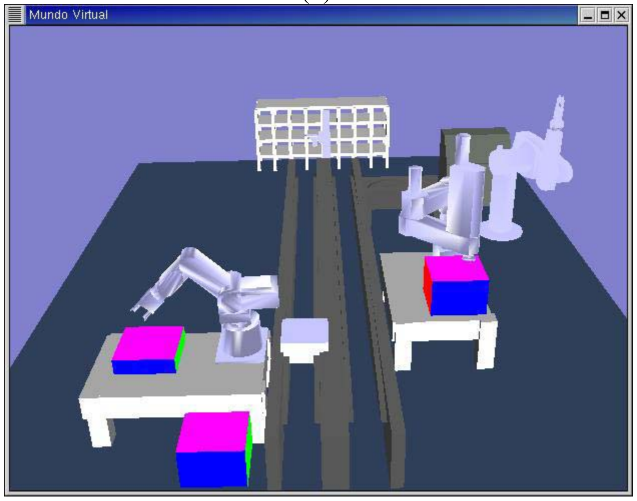
```
int a            //variable declaration
a = 0
move(2,600)     //go to the front
move(1, -1050) //go to the right
flag (1)      //Indicating Puma to begin
rcelflag(1,a) //Waiting for the Puma
while(a == 0){
      wait(500)
      rcelflag(1,a)
}
```



(a)



(b)



(c)

Figure 5. Snapshots taken during the practice execution.
(a) Puma working with the piece.
(b) Amatrol Jupiter stamping.
(c) Mitsubishi beginning its work.

In figure 5, we show s set of pictures taken during the execution of the defined task.

This practice session has been tested using a three participant simulation. Manipulating several robots at the same time during the execution, does not affect the response of the visual simulation. However, several tests have to be done in order to determine how the network latency affects the performance of the system.

Only four students that provide some suggestions and changes have tested the system. In order to probe the feasibility of the system, a test with a group of students in Robotics Courses has to be done as a future work.

7. Conclusions

Using this robotics virtual laboratory is an option for creating practice sessions that would help in distance learning courses. The system presented is a visual simulation in which can be used for training without having risks or causing damage in the equipment. The net-VE architecture allows the collaboration between users during the execution of the practice and remote access to a simulation. The language defined for use in the system allows programming the robots and there exists the possibility of defining task as a set of programs.

8. Future Work

There are many things that could be improved. The programming language can be extended in order to permit task configuration without using the current interface. We are also considering the possibility of adding the use of maps in the language that could be used to simulate sensors for the robots. The scripts used to define a simple assistant must be generalized in order to support a more complex assistant and possibly the ability of give explanations. The process of defining new practice sessions could be improved adding an interface to introduce the instructions without using a text file.

9. Acknowledgements

The work described in this paper is part of the project “Virtual Laboratories” granted by REDII- CONACyT.

10. References

[1] J. Sánchez, F. Morilla, et al. “Virtual and Remote Control Labs Using Java: A Qualitative Approach”. *IEEE Control Magazine*, April 2002.

[2] M. Gertz, D. Stewart, et al. "A Human-Machine Interface for Distributed Virtual Laboratories". In *IEEE Robotics & Automation Magazine.* December, 1994.

[3] S. Singhal, et. al., *Networked Virtual Environments, Design and Implementation*, ACM Press, 1999.

[4] T. Capin et. al. *Avatars in Networked Virtual Environments*, John Wiley & Sons, LTD, 1999.

[5] C. Bouras and G. Horni. "Architectures supporting e-Learning through Collaborative Virtual Environments: The case of INVITE". In *IEEE International Conference on Advanced Learning Technologies*, Madison, WI, USA, 2001.

[6] I. Rudomín, L. Muñoz, and M. Alencastre. "Virtual Laboratory". In *MICAI/TAINA,* Mérida, México, 2002.

[7] A.C. Catlin, et. al., "SoftLab : A virtual laboratory framework for computacional science", Technical Report, 1997.

[8] S. Chakaven, et. al., "DELTA´s Virtual Physics Laboratory. A Comprehensive Learning Plataform on Physics & Astronomy". In *Visualization*, San Francisco, CA, USA, 1999.

[9] K.W. Cheng, et. al. "Virtual Laboratory Development for Teaching Power Electronics". In *Power Electronics Specialists Conference*, Cairns, Australia, 2002.

[10] A. Speck and H. Klaeren, "RoboSiM: Java 3D Robot Visualization", In *Proceeding of 25th annual Conference of the IEEE (IECON)*, San José, CA, USA, 1999, Vol. 2, p. 821-826.

[11] O. Michel, P. Saucy, et. al. "KhepOnTheWeb: An Experimental Demonstrator in Telerobotics and Virtual Reality". In *Proceeding of International Conference on Virtual Systems and MultiMedia (VSMM)*, Geneva, Switzerland, 1997.

[12] L. Queiroz, M. Bergerman, et. al. "A Robotics and Computer Vision Virtual Laboratory". In *AMC*, Coimbra, Portugal, 1998, p. 694-699.

[13] L. Jin and I. Oraifige, "E-manufacturing in Networked Virtual Environments". In *IEEE, International Conference on Systems, Man and Cybernetics*, Tucson, AZ, USA, 2001.

[14] D. Gracanin, M. Matijasevic, et. al. "Virtual Reality Testbed for Mobile Robots". In *ISIE Bled*, Slovenia, 1999.

[15]JESS HomePage, http://herzberg.ca.sandia.gov/jess/, Last access December, 2002.