EJS+EJSRL: A FREE JAVA TOOL FOR ADVANCED ROBOTICS SIMULATION AND COMPUTER VISION PROCESSING

Carlos A. Jara, Francisco A. Candelas, Jorge Pomares, Pablo Gil and Fernando Torres
Physics, Systems Engineering and Signal Theory Department, University of Alicante, Spain

Keywords: Modelling, Robotics simulation, Virtual reality.

Abstract: This paper presents a free Java software platform which enables users to easily create advanced robotic applications together with image processing. This novel tool is composed of two layers: 1) Easy Java Simulations (EJS), an open-source tool which provides support for creating applications with a full 2D/3D interactive graphical interface, and 2) EjsRL, a high-level Java library specifically designed for EJS which provides a complete functional framework for modeling of arbitrary serial-link manipulators and computer vision algorithms. The combination of both components sets up a software architecture which contains a high number of functionalities in the same platform to develop complex simulations in robotics and computer vision fields.

1 INTRODUCTION

Robotics and Computer Vision (R&CV) systems have highly complex behaviours. For this reason, throughout the last two decades there has been a strong development of simulation tools devoted to R&CV systems. Some of these tools have been designed for professional applications, while others for educational and research purposes. In the field of industrial Robotics, several graphical software environments as for example Easy-ROB3D (Easy-ROB3D, 2004), have been created in the form of stand-alone business packages for well defined problems. These are powerful tools, but some of them lack of resources in some aspects for higher education. Otherwise, numerous open-source tools such as GraspIt (Pelossoft et. al, 2004), RoboMosp (Jaramillo et al., 2006) and Microsoft Robotics Studio (Jackson, 2007), overcome these deficiencies. Other open-source tools are in the form of toolboxes such as SimMechanics (SMC) (Babuska, 2005), RobotiCad (RBC) (Falconi and Melchiorri, 2008) and Robotics Toolbox for Matlab (Corke, 1996). With regard to Computer Vision tools, several libraries have been developed for education and research, such as the Open Computer Vision Library (OpenCV, 2001) and VXL (VXL, 2001), developed in C++ language, and Java Advanced Imaging (JAI, 2004), written in Java.

However, the majority of the above commented tools are independent software platforms which have been developed in a separated way. This feature represents a drawback when time comes to develop complex models which combine R&CV systems. Perhaps, only Robotics/Vision Matlab toolboxes provide a set of functions suitable for synthesis and simulation which can be programmed under the same environment. Nevertheless, both toolboxes do not provide a user-friendly graphical interface support for both creating a personalized application and building 3D virtual environments. Thus, educators and researches have to spend time and effort searching the suitable libraries and they must have programming skills to develop the application.

The approach presented in this paper is a new tool called EJS+EjsRL, which provides a complete functional framework for modeling and simulation of R&CV systems, all embedded in the same toolbox. In addition, this software platform gives full 2D and 3D graphical support both for creating user interfaces and complex robotic environments with computer vision algorithms in an easy and simplified way. The main novel feature of this approach is that its software architecture contains a higher number of functionalities in the same platform than the existing software applications for that purpose (see table 1). Most of these functionalities are included as high-level tools, with the advantage of allowing users to easily create R&CV applications with a minimum of programming. The tool presented contains several

A. Jara C., A. Candelas F., Pomares J., Gil P. and Torres F. (2010).
EJS+EJSRL: A FREE JAVA TOOL FOR ADVANCED ROBOTICS SIMULATION AND COMPUTER VISION PROCESSING.
In *Proceedings of the 7th International Conference on Informatics in Control, Automation and Robotics*, pages 153-160

features for Robotics such as kinematics, programming, dynamics, world modeling, importation of 3D model files, etc. and a higher number of Computer Vision algorithms than the JAI.

Table 1: Feature comparison with other toolboxes.

Feature	SMC	RBC	Matlab toolbox	EJS+ EjsRL
Kinematics	●	●	●	●
Dynamics	●	●	●	●
Programming		●	●	●
Importation VRML/OBJ			●	●
World Modeling				●
Computer Vision			●	●
Interface design				●
Software connection			●	●

Another meaningful problem is the platform dependency. Some C++ tools are not portable for all the operating systems. EJS+EjsRL is based on Java, a well-known programming language which is platform independent.

The remainder of this paper is organized as follows: section 2 describes the overall software architecture of the platform. Section 3 shows a complete application design. Section 4 shows other advanced features of the system. Section 5 shows the simulation capabilities of EJS+EjsRL by means of several test cases. Finally, some conclusions are discussed in section 6.

2 SYSTEM DESCRIPTION

2.1 Components

There are two main blocks that represent the functional core of this software platform: an object-oriented Java library (EjsRL) which allows users to model both arbitrary serial-link robots and computer vision algorithms, and *Easy Java Simulations* (EJS), powerful software for developing simulations. The combination of both tools (EJS+EjsRL) permits to easily and quickly create R&CV simulations.

EJS is a freeware, open-source tool developed in Java, specifically created for the creation of interactive dynamic simulations with higher graphical support (Esquembre, 2004). EJS has been designed for people who do not need complex programming skills. In order to develop a simulation, the user only provides the most relevant core of the algorithm and EJS automatically generates all the Java code needed to create a complete interactive application. There are a lot of applications which have been developed with EJS for research and teaching activities (Jara et al., 2008; Jara et al., 2009).

EjsRL is a Java library specifically designed for EJS which provides a complete functional framework that enables it to model and design advanced R&CV applications. All the components belonging to this software layer have been structured and organized in an object-oriented form. Figure 1 shows a simplified class diagram of EjsRL, specifying the most important packages and classes. For a complete description of all the classes, readers can visit the web page: http://www.aurova.ua.es/rcv.

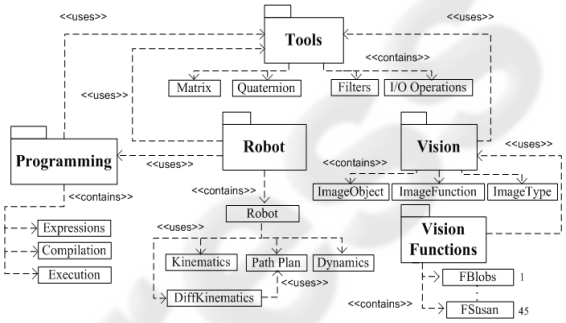


Figure 1: Package and class diagram of EjsRL.

2.2 Software Architecture

The software design is based on a hierarchical coordination between EJS and EjsRL. Each of them is divided into systems which must interchange data in order to develop R&CV simulations (figure 2).

A specific simulation within the EJS' environment must include the definition of the model and the definition of the view or graphical interface (figure 2). In order to describe the model, users must write the differential equations that establish how these variables change in time. For this last step, EJS offers two options. The first is a built-in editor of Ordinary Differential Equations (ODEs) in which users write the system equations in a similar way to how they would write on a blackboard. Users can choose different standard algorithms to numerically solve them (Euler, Runge-Kutta, etc.). The second facility is a connection with Matlab/Simulink that lets users to model systems with the help of these tools (Sanchez et al., 2005) (see section 5). In relation to the view, EJS provides a set of standard Java Swing, Java 2D and Java 3D components to build the interface in a simple drag-and-drop way. In addition, VRML and OBJ extern graphic files can be imported to the view. These

graphical components have certain properties that the user can connect with the model variables and set a link between the model and the view. Therefore, the simulation turns into an interactive application where users can change the model variables and observe the simulation behaviour in the view.

There are three important modules which define the most high-level API of EjsRL (figure 2): Robotics, Matrix Computation and Computer Vision. This library works as an external interface to give to model variables of EJS the corresponding value in order to create R&CV applications.

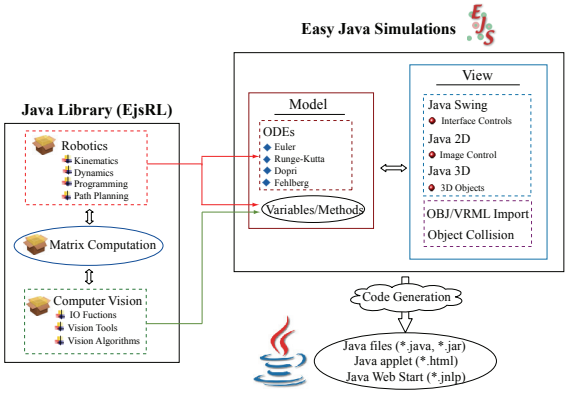


Figure 2: Software architecture of EJS+EjsRL.

3 DESIGN OF A ROBOTICS APPLICATION

3.1 Creating the Robot Arm and its Workspace

The first step in order to create a robotic simulation is to execute EJS and to insert the library EjsRL as external resource (figure 3). In this way, all the methods and classes of EjsRL can be used within EJS' environment. Secondly, it is necessary to create a specific robot in the model part. This action implicates to define the variables and to program a robot object specifying a minimum code.

For creating an arbitrary robot arm object, users only have to know its Denavit-Hartenberg parameters, its physical features and the type of joints. With these data, a Java object variable defined in the EJS' environment has to be initialized using the robot's constructor of the Robotics module of EjsRL. Figure 4 shows the Java code which must be inserted in the model of EJS'

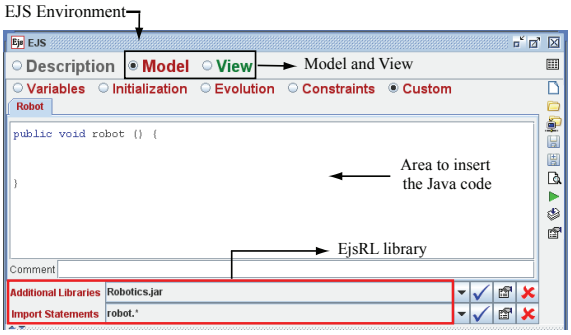


Figure 3: Overview of the EJS' environment

environment and the necessary variables to program a robot of 6 rotational DOF.

```
//DENAVIT-HARTENBERG variables
/*Fist Column (q)*/          /*Second Column (d)*/  /*Third Column (a)*/   /*Fouth Column (alpha)*/
DHParams[0] = q[0]+Math.PI/2;  DHParams[6] = l1;    DHParams[12] = 0;    DHParams[18] = -Math.PI/2;
DHParams[1] = q[1]-Math.PI/2;  DHParams[7] = 0;     DHParams[13] = a2;   DHParams[19] = 0;
DHParams[2] = q[2]+Math.PI;    DHParams[8] = 0;     DHParams[14] = 0;    DHParams[20] = Math.PI/2;
DHParams[3] = q[3];            DHParams[9] = l4;    DHParams[15] = 0;    DHParams[21] = -Math.PI/2;
DHParams[4] = q[4];            DHParams[10] = 0;    DHParams[16] = 0;    DHParams[22] = Math.PI/2;
DHParams[5] = q[5];            DHParams[11] = l6;   DHParams[17] = 0;    DHParams[23] = 0;

//DHq. Variables that change in the DH table
DHq[0] = 0;DHq[1] = 1;DHq[2] = 2;DHq[3] = 3;
DHq[4] = 4; DHq[5] = 5;

//Geometry
geometry[0]=l1;geometry[1]=a2;geometry[2]=0;
geometry[3]=l4;geometry[4]=0;geometry[5]=l6;

//Robot
char[] type = {'R','R','R','R','R','R'};
robot = new Robot(6, type);
robot.setDHParams(DHParams,DHq);
robot.setGeometry(geometry);
robot.setQLimits(qLimits);
robot.setInitPos(1,1,0);
```

EJS VARIABLES

Name	Value	Type	Dimension
robot	null	Object	
dof	6	int	
q	0	double	[dof]
l1	0.5	double	
l4	0.8	double	
l6	0.3	double	
a2	0.7	double	
qLimits	{-180,180,-180..	double	[dof*2]
DHParams	0.0	double	[dof*4]
DHq	0.0	int	[dof]
geometry	0.0	double	[dof]

Figure 4: Java code for modeling a 6 rotational DOF robot.

After programming the robot object, the next step is to develop the interface or view for the final user. As stated, EJS provides a set of components to build the interface in a simple drag-and-drop way. In the case of a robotic simulation, the interface can be composed by the 3D solid links of the robot and its workspace, and other standard components to control the application (panels, buttons, sliders, plots, etc.). Figure 5 shows the construction of the interface for the example proposed. The component *drawingPanel3D* is the 3D environment where the robot and its workspace will be displayed. Here, it is defined each one of the 3D links of the robot by means of the VRML component, which allows to import models from existent VRML files. As mentioned, all the interface components of EJS have certain properties which are used for the simulation. Figure 5 shows the properties of the VRML component (*Position and Size*, *Visibility and Interaction* and *Graphical Aspect*). The position and transform fields will be used to move the robot since they will be connected with the model variables which define the robot. Figure 5 also shows a dialog

(*Move Joints*) where some sliders controls (*q1…q6*) have been added from the view components. These sliders are connected with the *q* variable of the robot model (see figure 4).

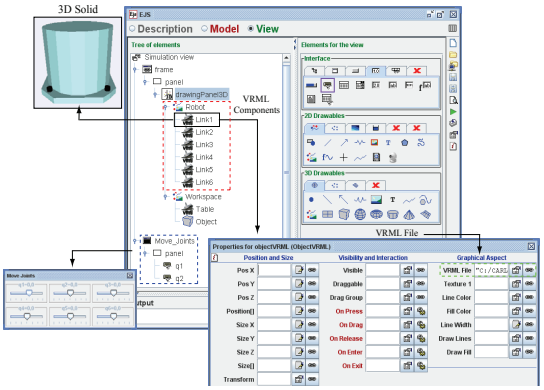


Figure 5: Interface construction of a Robotics application.

3.2 Kinematics and Path Planning Simulation

The implementation of the forward kinematics can be easily programmed and simulated with EJS+EjsRL. Figure 6 shows the Java code to resolve the forward kinematics of the robot proposed. The joint values are got from the interface Move Joints for updating the *DHParams* array of the model. Afterwards, the homogeneous transformations of each link are computed using the method *FKinematics* of the Robotics module of EjsRL. Finally, these matrix objects (*A01…A06*) are inserted in the property *Transform* (figure 5) of the VRML components in order to move them according this kinematics algorithm.

```
public void forwardKinematics () {
  //Update the current values of the joints
  DHParams[0] = q[0] + Math.PI/2;
  DHParams[1] = q[1]-Math.PI/2;
  DHParams[2] = q[2]+Math.PI;
  DHParams[3] = q[3];
  DHParams[4] = q[4];
  DHParams[5] = q[5];
```

q = (-47.6,...,-16.8)

Interface *Move Joints*

Virtual Environment

```
  //Compute the forward kinematics
  A01 = robot.FKinematics(DHParams[0],1);
  A02 = robot.FKinematics(DHParams[1],2);
  A03 = robot.FKinematics(DHParams[2],3);
  A04 = robot.FKinematics(DHParams[3],4);
  A05 = robot.FKinematics(DHParams[4],5);
  A06 = robot.FKinematics(DHParams[5],6);
}
```

A01..A06

Figure 6: Java code for the forward kinematics of a 6 rotational DOF robot.

EjsRL contains some methods to solve the inverse kinematics problem. Figure 7 shows an example for solving this based on the Jacobian operator. The method *IKinematics* receives the position and orientation of the end effector (Matrix *T*) and the current joint values of the robot (array *q_current*) as input parameters. Finally, the robot is moved to the suitable position using the forward kinematics method described before.

```
public void inverseKinematics () {
  //Current values of vector q
  double[] q_current = {q[0],q[1],q[2],q[3],q[4], q[5]};
  //Position and orientation of the end effector (X, Y, Z, Xº, Yº, Zº)
  Matrix T = new Matrix(4,4);
  T.set(0,3,X); T.set(1,3,Y); T.set(2,3,Z);T.set(3,3,1.0); //Position
  T.setMatrix(0,2,0,2,Maths.transRPYtoR(Roll, Pitch, Yaw)); //Orientation
  //Call to the inverse kinematics algorithm
  Solution sol = robot.IKinematics(T,q_current);
  if(sol!=null){
      q[0] = sol.getElemSolution(0); q[1] = sol.getElemSolution(1);
      q[2] = sol.getElemSolution(2); q[3] = sol.getElemSolution(3);
      q[4] = sol.getElemSolution(4); q[5] = sol.getElemSolution(5);
      //Move the robot with the updated q values
      forwardKinematics();
  }
}
```

Figure 7: Java code for the inverse kinematics problem.

With regard to trajectory planning, EJS+EjsRL allows users to easily perform the simulation of many path planning algorithms for n-axis robot arms. The ODEs editor implemented in EJS is employed to generate the position, velocity and acceleration values. The Robotics classes of EjsRL contain a path planning module which computes the acceleration parameters of several trajectories from their imposed constrains. Thus, two steps are only necessary to create a planning algorithm for a n-axis robot manipulator:

- To write the equations of the basic motion of a multi-body system. Figure 8 shows these equations in the ODEs editor of EJS. These equations compute the sequence values of the position (*q*) and velocity (*VPlan*) of all the robot joints from the acceleration of the trajectory (*APlan*);
- To compute the acceleration of the path planning algorithm proposed using one of the functions provided by the Robotics package. The trajectory planning module returns the acceleration parameters of several kinds of trajectories which can be used in the motion equations;

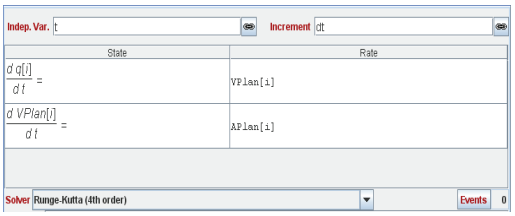


Figure 8: ODEs of basic robot motion.

There are a lot of methods implemented in the Robotics module: splines, cubic interpolators, synchronous, asynchronous and linear trajectories, and the 4-3-4 polynomial path planning algorithm. Figure 9 shows the Java code to program this last interpolator in order to determinate the acceleration array for the differential equations.

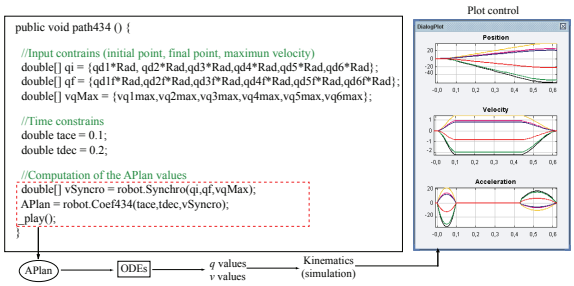


Figure 9: Simulation of a 4-3-4 polynomial trajectory.

The generated joint values are automatically given to the kinematics model to simulate the robot movement (method *_play*). In addition, EJS plot controls can be used to visualize the trajectory variables (figure 9).

3.3 Dynamics Features

The Robotics module of EjsRL implements numerical methods to solve the forward and inverse dynamics problems (Newton-Euler and Walker-Orin, respectively). Figure 10 shows an example which obtains the inverse dynamics with an external force. Mass, inertias and friction properties must be known in order to solve this algorithm. The array variables *VPlan* and *APlan* belong to the velocity and acceleration of the path planning previously computed.

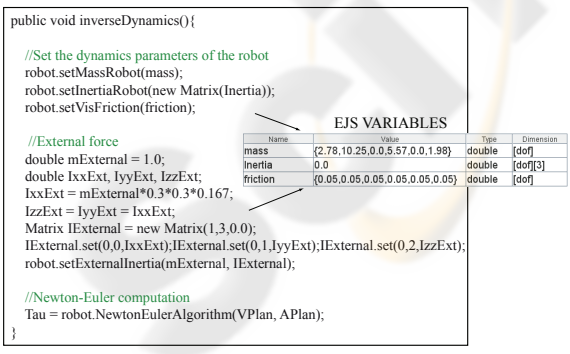


Figure 10: Programming the inverse dynamics with an external force.

3.4 Using Computer Vision Features

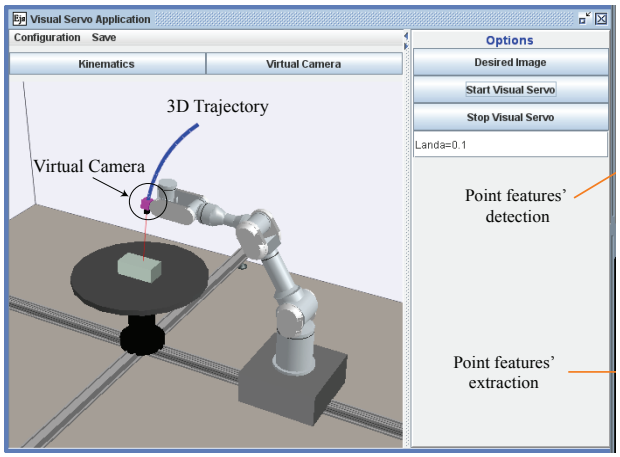
The Computer Vision classes of EjsRL provide a complete library for the development of image processing algorithms within EJS' environment. There are approximately fifty different functions implemented in this module, ranging from basic operations (format conversion, image adjustment, histogram, etc.) to image feature extraction (point and edge features). As example, authors implement a computer vision algorithm in the virtual robotic environment previously created. The aim is to perform an Eye-In-Hand (EIH) vision based control using four corner features in the control loop.

First of all, it is necessary to obtain a view projection from the end effector of the robot. For that end, EJS has an option which allows users to create a virtual camera in the 3D robotic environment. Figure 12 shows the appearance of the interface developed where the window "Virtual Camera" shows the projection of the EIH virtual camera. Secondly, this projection must be processed in order to extract the corner features of the object. Figure 11 shows the Java code which computes corner detection in the virtual camera's image (this code can also be used for real images). Initially, the image of the virtual camera control is obtained (variable *vcamera*) and the image objects are created. Afterwards, the processing algorithm is defined by means of the *ImageFunction* interface. Finally, the image is processed (*processImag* method), the point features are detected using one of the implemented algorithms, for example the SUSAN method (Smith and Brady, 1997), and these are returned as an array variable. These point features can be seen in the window "Virtual Image" of the figure 12.

```
public ArrayList Corner_Detection(){

  //Get the Virtual Camera component
  ControlElement vcamera = _view.getElement("virtualCamera");

  //Create the image objects
  ImageObject initial_Image = new ImageObject(vcamera.getImage());
  ImageObject result_Image = null;

  //Proccess the image and extract the point features
  ImageFunction f1 = new FColorToGray(); //Function Color_to_Gray
  ImageFunction f2 = new FSusan();       //Function Corner detector
  result_Image = new ImageObject(f2.processImg (f1.processImg(initialImage)));
  ArrayList pointsSusan = ((FSusan)f2).getPoints();
  return pointsSusan;
}
```

Figure 11: Java code for detecting corner features in the virtual image.

The control action and the interaction matrix used in this control algorithm are based on a


Visual Servo Application
Configuration
Save
Kinematics
Virtual Camera
3D Trajectory
Virtual Camera
Options
Desired Image
Start Visual Servo
Stop Visual Servo
Landa=0.1
Point features' detection
Point features' extraction

system, called RobUALab.ejs (Jara et al., 2008), allows users to simulate path planning algorithms in a virtual robotic environment, as well as execute remote commands in a real robotic plant.

Programming classes of EjsRL (figure 1, package "Programming") enable users to develop Java routines in a robotic simulation. Figure 14 shows a programming experiment which consists of doing pick-and-place operations of virtual objects located in the conveyor belt using synchronous trajectories (parameter "Syn" in the method *moveJ*).

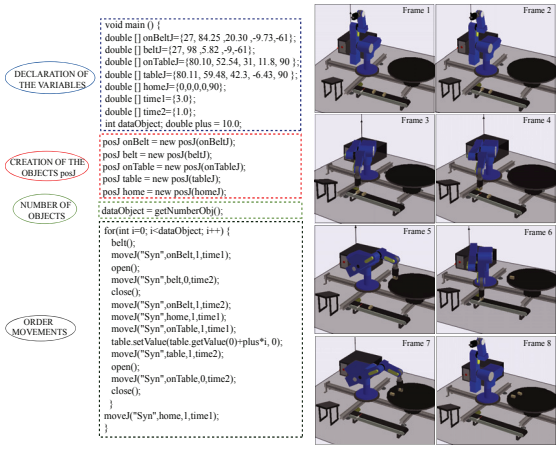


Figure 14: States of the virtual robot during the execution of the programming experiment.

5.2 A Multi-robot System

EjsRL allows users the instantiation of different robot objects. Thus, it is possible to developed multi-robot simulations in an easy way. As example, a multi-robotic system composed by a PA-10 robot of 7 rotational DOF and a 3 rotational DOF robot (RRR) is presented here. This last serial robot is attached to a link of the upper part of the PA-10 (figure 15). In addition, the robot RRR has a virtual camera at the end as an EIH configuration. Figure 15 shows the interface of the application developed: on the left, the 3D virtual environment of the workspace; on the right, the virtual projection of the EIH camera located at RRR.

6 CONCLUSIONS

In this paper, a free Java-based software platform for the creation of advanced R&CV applications has been presented. EJS+EjsRL is a suitable tool to develop research and educational simulations in R&CV systems. The paper has showed several high-level applications which illustrate a part of the possibilities of EJS+EjsRL. More information can be obtained from http://www.aurova.ua.es/rcv, where readers can also execute a lot of test examples.

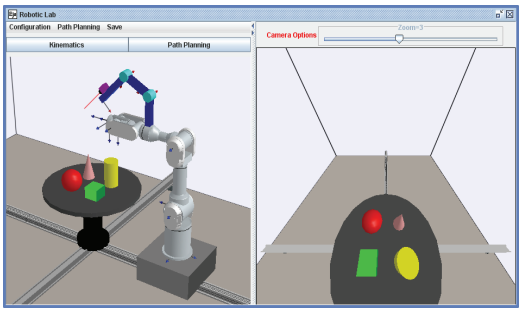


Figure 15: Appearance of the interface of a multi-robot simulation.

ACKNOWLEDGEMENTS

The work presented in this paper is supported by the Spanish Ministry of Education and Science (MEC) through the research project DPI2008-02647.

REFERENCES

Babuska, R. (2005). Design Environment for Robotic Manipulators. In *Proceedings of the 16th IFAC World Congress,* vol. 16, Prague.

Corke, P. (1996). A Robotics Toolbox for MATLAB. *IEEE Robotics and Automation Magazine*, 3(1): 24-32.

Easy-ROB Software (2004). http://www.easy-rob.com/.

Esquembre, F. (2004). Easy Java Simulations: A software tool to create scientific simulations in Java. *Computer Physics Communications,* 156(2): 199-204.

Falconi, R. and Melchiorri, C. (2008). Roboticad: An Educational Tool for Robotics. In *Proceedings of the 17th IFAC World Congress*, vol. 17, Seoul.

Gourdeau, R. (1997). Object-oriented programming for robotic manipulator simulation. *IEEE Robotics & Automation Magazine,* 4(1): 21-29.

Jackson, J. (2007). Microsoft Robotics studio: A technical introduction. *IEEE Robotics & Automation Magazine,* 14(1): 82-87.

Jara, C., Candelas, F. and Torres, F. (2008). An advanced interactive interface for Robotics e-learning. *International Journal of On-line Engineering*, 4(1): 17-25.

Jara, C., Candelas F., Torres, F., Esquembre, F., Dormido, S. and Reinoso, O. (2009). Real-time collaboration of virtual laboratories through the Internet. *Computers & Education*, 52(1): 126-140.

Jaramillo, A., Matta, A., Correa, F. and Perea, W. (2006). ROBOMOSP. *IEEE Robotics & Automation Magazine,* 13(1): 62-73.

Java Advanced Imaging library (JAI) (2004). http://java.sun.com/products/javamedia/jai.

Open Source Computer Vision Library (2001). http://www.intel.com/research/mrl/research/opencv.

Pelossof, R., Miller, A., Allen, P and Jebara, T. (2004). An SVM learning approach to robotic grasping. In *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 21, Taiwan.

Sanchez, J., Esquembre, F., Martín, C., Dormido, S., Dormido-Canto, S., Canto, R., Pastor, R. and Urquia, A. (2005). Easy Java Simulations: an open-source tool to develop interactive virtual laboratories using MATLAB/Simulink. *International Journal of Engineering Education*, 21(5): 789-813.

Smith, S. and Brady, J. (1997). SUSAN: a new approach to low level image processing. *International Journal of Computer Vision*, 23(1): 45-78.

VXL libraries (2001). http://vxl.sourceforge.net/.