EJS+EjsRL: an interactive tool for industrial robots simulation, computer vision and remote operation

Abstract

This paper presents an interactive Java software platform which enables users to easily create advanced robotic applications together with computer vision 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 and simulation of arbitrary serial-link manipulators, computer vision algorithms and remote operation. 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. In addition, the paper shows its successful application to virtual and remote laboratories, web-based resources that enhance the accessibility of experimental setups for education and research.

***Keywords**: modeling, robot simulation, Robotics education, visualization tools*

1. Introduction

Robotics and Computer Vision (R&CV) systems have highly complex behaviours. The best and the most useful way to undertand how they work is through their modeling. Models are usually used to characterize and optimize their performance via numerical solutions, so the highly complexity of R&CV systems need a reliable model to enable users solving real problems by means of the simulation. Thus, the development of powerful computational platforms that allow to model and simulate their behaviour constitutes a fundamental tool for designers, users, and researchers of these fields.

From the last two decades to nowadays, 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, and more specifically for robot arms, several graphical software environments such as RoboWorks [1], Robot Assist [2] and Easy-ROB3D [3] have been created in the form of stand-alone business packages for well defined problems. These are powerful tools, but they lack of resources in some aspects for higher education and research. In this way, numerous open-source tools, overcome these deficiencies. Among them, it is worth pointing out RoboSim [4], GraspIt [5], RoboMosp [6] and Microsoft Robotics Studio [7]. Other open source packages are in the form of toolboxes. Outstanding examples of this kind of platforms are RoboOp [8], Form [9], Modelica [10] and ViSP [11], based on an object-oriented design, and SimMechanics [12], RobotiCad [13] and Robotics Toolbox [14], Matlab-Simulink computational libraries

With regard to Computer Vision processing tools, several libraries and systems have been developed for research and education. It is worth mentioning the Open Computer Vision Library (OpenCV) [15], VIGRA [16] and VXL [17], some of the most complete and efficient computer vision libraries develop in C++ language. Besides, there are a few libraries written in Java. Java Imaging Graphics Library (JIGL) [18] and Java Advanced Imaging (JAI) [19] from Sun Microsystems represent toolbox applications for standard image-handling algorithms.

However, the majority of the above commented tools are independent software platforms which have been created in a separated way: on the one hand, robotics tools and on the other hand, computer vision programs. This feature represents a drawback when time comes to develop complex models which combine R&CV systems. Perhaps, only Robotics/Vision

Matlab Toolboxes and ViSP provide a set of functions suitable for synthesis and simulation of both systems which can be programmed in an easy way under the same environment. Nevertheless, both toolboxes do not provide a user-friendly graphical interface support for both creating a personalized application and building solid 3D robot links together with their environment (external objects). This represents a drawback for educators and researches because they have to spend time and effort searching the suitable libraries and developing the application. In addition, they must have programming skills in order to develop the graphical interface with other external tools.

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 supports both for creating interfaces and complex robotic environments in an easy 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 end (Table I). Most of these functionalities are included as high-level tools, with the advantage of allowing users capabilities to easily create R&CV applications with a minimum programming.

Some aforementioned tools such as RoboMosp and Microsoft Robotics Studio (MRE), are powerful systems with contain a high number of functionalities. MRE is designed for specific robots (mainly for mobile robots) and programming an arbitrary serial link manipulator entails editing code and programming the Robotics algorithms with detail. RoboMosp is well oriented to Robotics simulation, but does not contain other capabilities (see Table 1). In addition, both programs do not provide computer vision support for image processing.

The work presented here contains several advanced features for Robotics. It covers functions for solving problems ranging from kinematics, trajectory planning, programming, dynamics, object interaction, world modeling, importation of 3D model files and so on. It has been included Internet communication methods via HTTPS to enable user create remote applications. There are a higher number of Computer Vision algorithms than the JIGL and JAI Java libraries. Moreover, the tasks of creating the user interface (windows, control buttons, sliders, etc.) and developing the virtual environment of the simulation (3D objects) are quite simplified. Thus, EJS+EjsRL has the advantage respect to other libraries that provides vision and robotic functions together, and the users can simulate robot movements and sensorize them without changing the programming platform. All this leads to the deduction that the presented approach is a powerful tool to create advanced Robotics applications together with Computer Vision processing.

Another meaningful problem is the platform dependency. Some C++ tools such as RoboOp, ViSP or MRE are not portable for all the operating systems. The tool presented is based on Java [20], an advanced programming language which is platform independent. User requires only installing the corresponding Java plug-ins in order to execute Java applications. Java features allow users to share its computational resources in distributed environments.

Features	SimMechanism	RoboMosp	MRE	Matlab Toolbox	EJS+EjsRL
Multiplatform support	●	●		●	●
Kinematics and Dynamics	●	●	●	●	●
Path Planning	●	●	●	●	●
Programming		●	●	●	●
Import VRML and OBJ		●	●	●	●
Collision Detection			●		●
World Modeling		●	●		●
Image Reading/Writing				●	●
Computer Vision functions				●	●
Remote Operation methods					●
Interface design			●		●
External software connection		●	●	●	●

Table 1. Feature comparison with other toolboxes

Summarizing, EJS+EjsRL presents a set of important features with regard to other existent tools such as: it integrates R&CV operations, it is easy to use and it does not require special programming skills for the final user. In addition, it is platform independent and it generates high quality graphical simulations.

The remainder of this paper is organized as follows: Section 2 describes the overall software architecture of the platform and the elements which compose it. Section 3 describes how to model and simulate a problem about a visual servo control application of a 6 rotational DOF robot. Afterwards, in Section 4, other features such as remote operation methods, external connection with other tools, augmented reality features and collision detection will be presented. Section 5 shows the simulation capabilities of EJS+EjsRL by means of several test cases. Section 6 describes a comparison with other tools in terms of accuracy and performance. Finally, the most important conclusions are discussed in Section 7.

2. System Overview

In this section, a detailed description of the software architecture will be presented. First, the main components which make up the tool will be detailed. Next, the software architecture design will be explained. Finally, some issues about the code generation will be shown.

2.1. Components

As mentioned before, there are two main blocks that represent the functional core of this software platform: an object-oriented Java library (EjsRL from this moment) which allows users to model both arbitrary serial-link robots and computer vision algorithms, and *Easy Java Simulations* (EJS), an open-source software which represents a powerful tool for easily developing simulations with a higher graphical interface capacity. The combination of both software tools (EJS+EjsRL) makes possible to easily and quickly create advanced R&CV simulations.

EJS is a freeware, open-source tool developed in Java, specifically created for the creation of interactive dynamic simulations [21]. EJS has been designed for people who do not need complex programming skills. Users can easily and quickly create interactive simulations. They need to provide only the most relevant core of the simulation algorithm and EJS automatically generates all the Java code needed to create a complete interactive simulation, including a wide range of sophisticated software techniques (such as handling computer graphic routines, communication protocols, multi-threading and others). EJS is totally implemented in Java language. This feature gives a full portability of the applications generated with this tool and they can be executed on different operating systems. In addition, external Java libraries can be used as auxiliary framework to develop the simulation code. This fact enables users to design more complex applications within the EJS environment, such as R&CV applications using the high-level library EjsRL. In short, EJS can be considered a powerful tool for easily developing simulations. Currently, there are a lot of applications which have been developed with this software for several science fields, research and teaching activities [22-25] and EJS has been employed to develop complex and advanced functionalities [26-28].

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

There are four important blocks or modules which define the most high-level API of the EjsRL (Fig. 1): Robotics, Matrix Computation, Computer Vision and Remote Operation. In addition to the aforementioned blocks, the library incorporates import/export functions for different file formats in order to allow users to save and restore their designs. In relation to these four main modules, their functional architectures are based on the following criteria:

- The Matrix Computation classes perform the mathematical computations required for solving all the R&CV algorithms. They cover the fundamental operations of numerical linear algebra, such as matrix and vector arithmetic (addition, multiplication, norms, etc).
- The Robotics block covers the fundamental engine for arbitrary serial-link Robotics computation such as kinematics, trajectory planning and dynamics. Most of the methods implemented are based in the well-known standard Robotics algorithms. In addition, an interpreter of Java code has been included to allow users the compilation and execution of programming routines for robots in this language.
- The Computer Vision classes contain a wide range of methods for image processing. It has implemented several standard Computer Vision algorithms which perform operations like image transform and conversion, image adjustments, convolution, etc. Moreover, it allows users to handle different types of images (binary, gray scale, colour) and it incorporates geometrical data manipulation (points, edges and segment).
- The Remote Operation functions permit users to communicate with remote devices using the HTTPS protocol. They are composed by several classes to perform HTTPS orders in an easy way.

2.2. Software architecture

The software design is based on a hierarchical coordination between EJS and EjsRL, the two high level layers of the software architecture. Each of them is divided into subsystems which must communicate and interchange data in order to develop advanced simulations (Fig. 1).

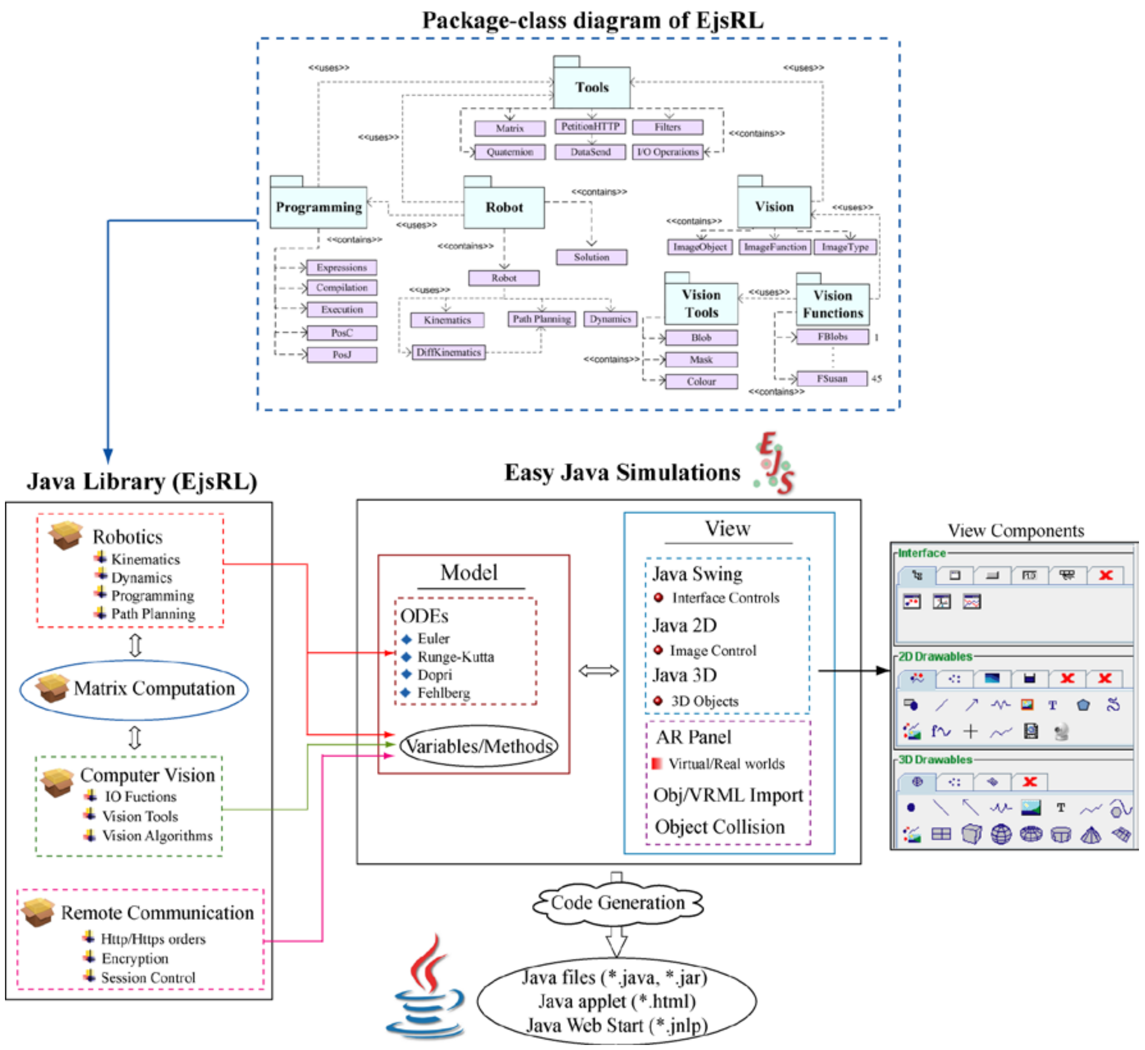


Figure 1. Software architecture of the tool and package-class diagram of EjsRL

In general terms, EJS provides a complete 2D/3D graphical interface support and model construction tools in order to easily build advanced applications. Furthermore, EjsRL gives all the R&CV algorithms which are necessary to simulate the system behaviour within an EJS application.

A specific simulation within the EJS' environment includes the definition of the model and the definition of the view or graphical interface (Fig. 1). Into the model, users can declare different types of variables (int, double, String, Object) in order to describe the system. Java Object variable represents an abstract wrapper to call external objects from other Java libraries. The approach presented in this paper utilizes this property to create "robots" and "images" objects of EjsRL to use them into the EJS' environment. In order to describe the model, users must write the differential equations that establish how these variables change in time or under user interaction. 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 (Euler–Richardson, Runge–Kutta, Fehlberg, etc.) to numerically solve them. The second facility is a connection with Matlab/Simulink that lets users specify and solve their models with the help of these tools [29]. This flexibility permits users to create advanced and complex simulations. This could be seen in Section 4.

In relation to the view, EJS provides a set of standard Java Swing (windows, panels, buttons, text files, string labels, sliders, combo-boxes, etc.), Java 2D (particles, polygons, vectors, planes, images, etc.) and Java 3D components (box, cone, cylinder, ellipsoid, plane, vectors, etc.) (see Fig. 2, view components) to build the interface in a simple drag-and-drop way. In addition, VRML, 3DS 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. Other important robotic features of EJS are the *Augmented Reality* panel control (AR panel), where 3D virtual worlds can be complemented with real information inside the same interface, and object collision detection (Section 4).

The library EjsRL works as an external software interface to give to model variables of EJS the corresponding value in order to create R&CV applications. Thus, a model variable from EJS' environment can be initialized with an Object instance of the Java library. This Object will contain embedded the R&CV engine and its methods will be able to be applied to the graphical components which will simulate the system behaviour. Therefore, creating an application that simulates a robot or a computer vision algorithm is greatly simplified.

2.3. Code generation

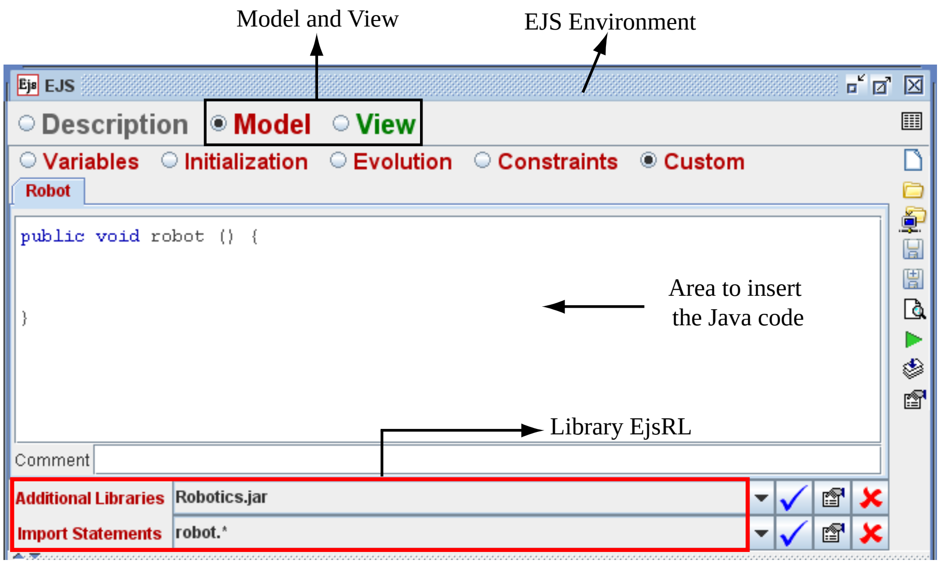
EJS automatically generates all the Java code needed to create a complete interactive simulation once users have finished the construction of the simulation (model and view). In addition, EJS creates all the necessary files in order to run the final applications in three different ways (Fig. 1): as an *applet* embedded in a web page (html file), as a stand-alone Java application (jar file), and as a Java Web Start application (jnlp file).

3. Using EJS+EjsRL: visual servo control simulation of a 6 DOF robot

This section describes how to easily design, model and simulate a visual servo control application of a 6 rotational DOF robot using the presented approach. The main requirements for the proposed problem is the implementation of a high part of the standard Robotics algorithms (kinematics, path planning and dynamics) adding an Eye-In-Hand (EIH) vision based control using corner features in the control loop to move the virtual robot previously created. This section shows an outline of the steps to carry out the development using EJS+EjsRL in order to get a solution for the simulation proposed.

3.1 Creating the 6 rotational DOF robot

The first step in order to create the required simulation is to execute EJS and to insert the library EjsRL as external resource (Fig. 2). In this way, all the Robotics algorithms of EjsRL can be used within EJS' environment for the simulation. Secondly, it is necessary to create the 6 DOF robot in the model part. This action implicates to define the variables and to program a robot object specifying a minimum code.

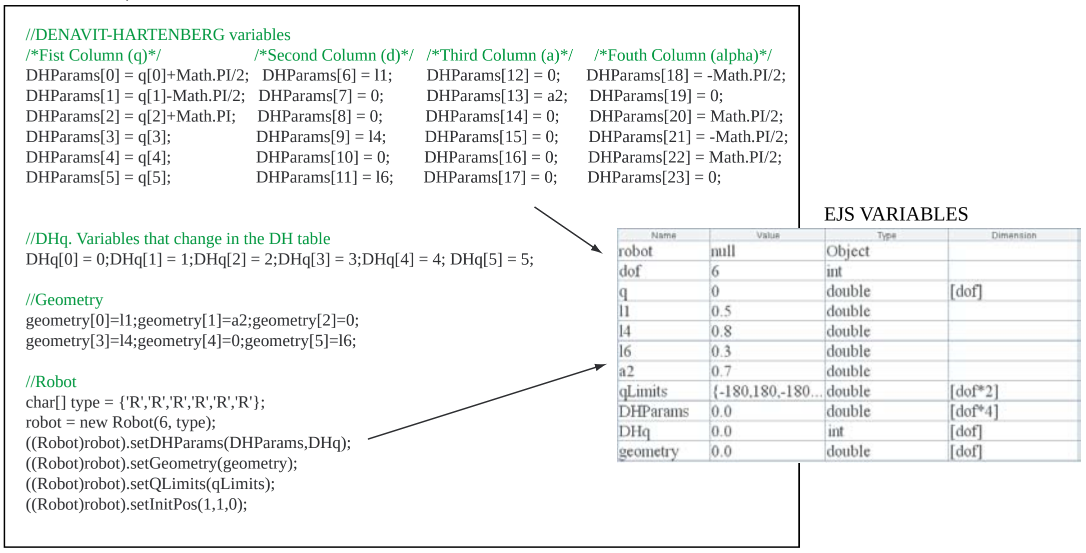


```
public void robot () {


}
```

Figure 2. Overview of the EJS' environment

For creating an arbitrary robot arm object, users only have to know its Denavit-Hartenberg (DH) parameters [30], its physical features, the type of joints (rotational or prismatic) and its range of motion (maximum and minimum values). 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. Fig. 3 shows the Java code which must be inserted in the model of EJS' environment (Fig. 2, "Area to insert the Java code") and the variables that must be created to program the robot arm of 6 rotational DOF. The parameter *type* indicates the kind of each joint (rotational 'R', prismatic 'P') for the robot's constructor.



```
//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)robot).setDHParams(DHParams,DHq);
((Robot)robot).setGeometry(geometry);
((Robot)robot).setQLimits(qLimits);
((Robot)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 3. Java code for the model in order to create a 6 rotational DOF robot.

After programming the robot object, the next step is to perform the interface or view. 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 construction of the robot and its workspace, and other standard components to control the application (windows, panels, buttons, sliders, plots, etc.). Fig. 4 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 commented before, all the interface components of EJS have certain properties which are used for the simulation. Fig. 4 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 (see subsection 3.2) since they will be connected with the model variables which define the robot arm.

Fig. 4 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. In this way, users will be able to move the robot specifying a joint position by means of this control dialog.

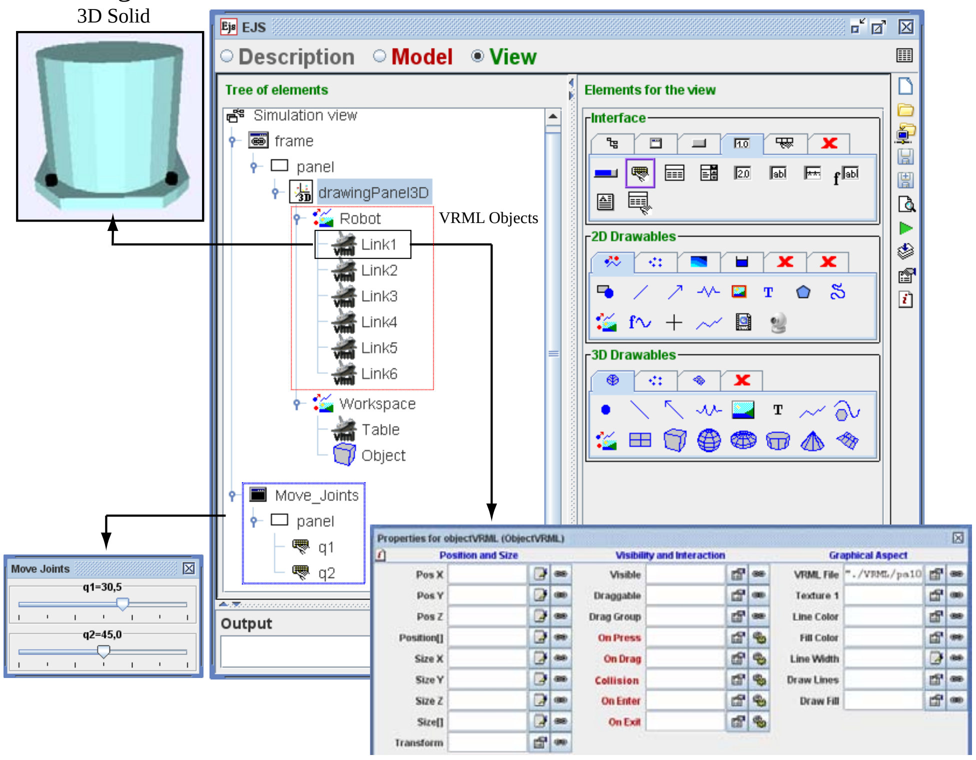


Figure 4. Interface construction of the Robotics application proposed

3.2 Kinematics simulation of the virtual robot

Robot kinematics deals with the analytical study of the motion of a manipulator. There are two well-known problems: the forward and inverse kinematics problems. The Robotics classes of EjsRL implement both algorithms in order to give motion to the 3D links of a specific manipulator defined in an EJS' view.

3.2.1. Forward Kinematics

The implementation of the forward kinematics problem is based on a sequential multiplication of the homogeneous transformations that describe the spatial relation between the joint values and the spatial location of the end effector. This algebraic algorithm can be easily programmed and simulated with EJS+EjsRL. Fig. 5 shows the Java code to resolve the forward kinematics computation of the robot considered. The joint values *q* are got from the dialog *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* (see Fig. 4) of the VRML components to move them according to this algorithm.

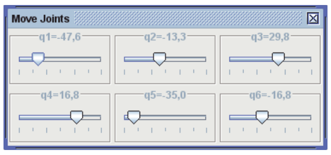
```
public void forwardKinematics () {

  //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];

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

q = (-45.6,..,-16.8)

Interface *Move Joints*



Move Joints

q1=-47,6 q2=-13,3 q3=29,8
q4=16,8 q5=-35,0 q6=-16,8

A01..A06

Virtual robot

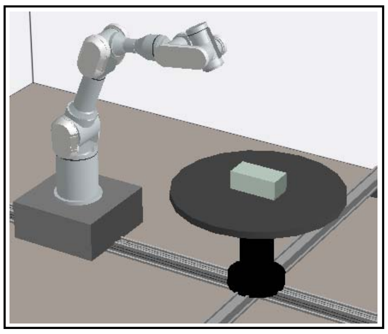


Figure 5. Java code for the forward kinematics of the 6 rotational DOF robot modelled with VRML objects

3.2.2. Inverse Kinematics

The numerical solution developed in EjsRL for the inverse kinematics problem is based on the robot Jacobian operator. This algorithm is an iterative procedure where the initial position affects both the search time and the solution found. In addition, some solutions could not be possible if the joint values computed describe an end-point out of reach of the manipulator.

Fig. 6 shows the Java code for the inverse kinematics implementation of the robot proposed and its simulation in the virtual environment. 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. The method also checks that the solution proposed is inside the area of reach of the manipulator. Finally, the robot is moved to the suitable position using the function for forward kinematics explained before.

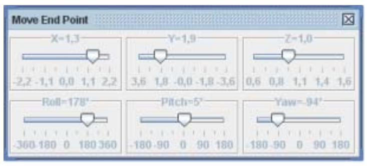
```
public void inverseKinematics () {

  //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,Roll°,Pitch°,Yaw°)
 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)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();
 }
}
```

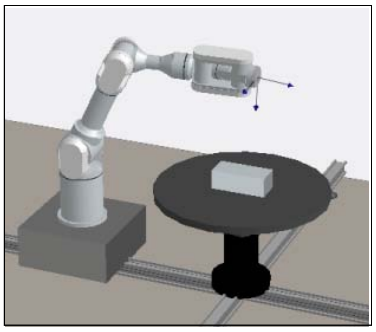


Move End Point

X=1,3 Y=1,9 Z=1,0
Roll=178° Pitch=5° Yaw=-94°

X,Y,Z,Roll,Pitch,Yaw

Virtual robot



q = (85.09,..,-86.54)

Figure 6. Java code for the inverse kinematics implementation

3.3 Path planning implementation

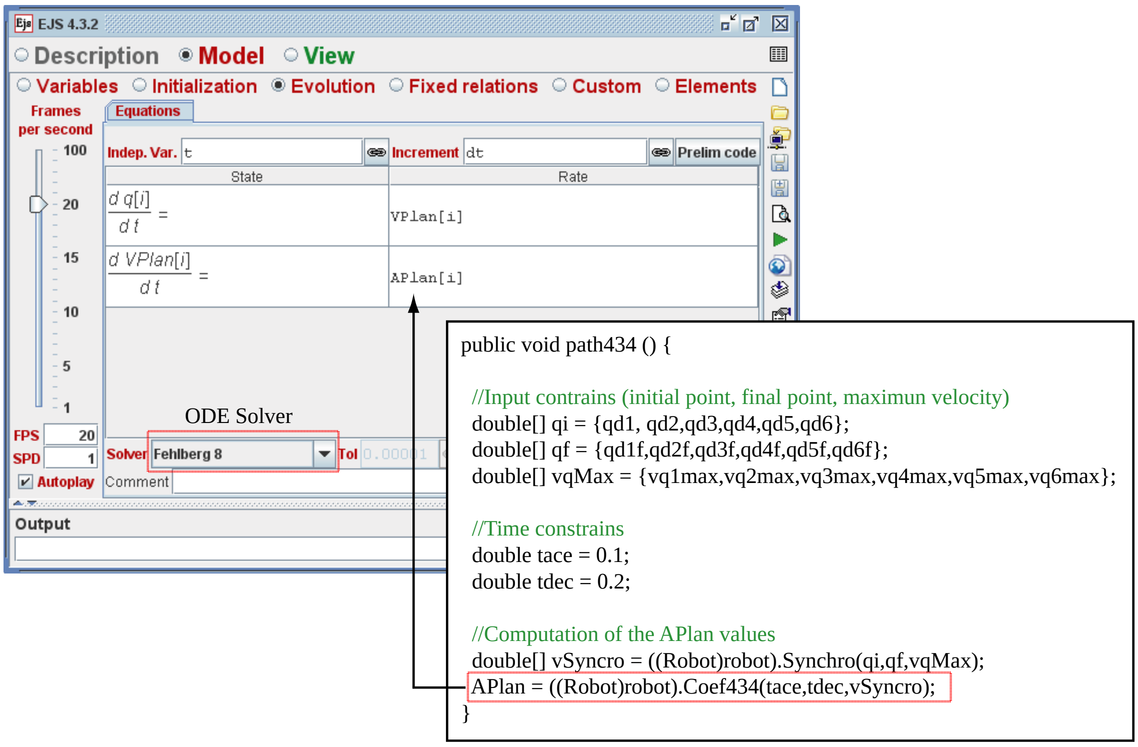
EJS+EjsRL allows users to easily perform the simulation of path planning trajectories for n-axis robot arms in both Cartesian and joint spaces. On the one hand, the built-in ODEs editor implemented in EJS is employed to generate the position, velocity and acceleration values. This feature allows users to write differential equations to model the problem. On the other hand, the Robotics classes of EjsRL contain a path planning construction module which computes the acceleration parameters of several trajectories from their imposed constrains.

Two steps are only necessary in order to create a path planning algorithm for the 6 DOF robot proposed:

1. To write the differential equations of the basic motion of a multi-body system. These are the typical equations for the description of a movement in space. Fig. 7 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*).

2. 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 trajectories which will be used in the motion equations.

There are a lot of methods implemented in the Robotics module for the simulation of the robot motion. It can be mentioned splines (third, fourth and fifth order), cubic interpolators, synchronous, asynchronous and linear trajectories, and the 4-3-4 polynomial path planning algorithm. Fig.7 shows the Java code to program this interpolator in order to determinate the acceleration array for the differential equations of motion. The joint values generated are given to the kinematics model to simulate the robot movement as it is shown in Fig. 8, where some EJS plot controls have been used to visualize the more interesting variables of the trajectory.



Indep. Var. t   Increment dt

$$\frac{d\,q[i]}{dt} = \texttt{VPlan[i]}$$

$$\frac{d\,VPlan[i]}{dt} = \texttt{APlan[i]}$$

FPS 20 SPD 1 Autoplay [✓]

Solver Fehlberg 8 Tol 0.00001

```
public void path434 () {

  //Input contrains (initial point, final point, maximun velocity)
  double[] qi = {qd1, qd2,qd3,qd4,qd5,qd6};
  double[] qf = {qd1f,qd2f,qd3f,qd4f,qd5f,qd6f};
  double[] vqMax = {vq1max,vq2max,vq3max,vq4max,vq5max,vq6max};

  //Time constrains
  double tace = 0.1;
  double tdec = 0.2;

  //Computation of the APlan values
  double[] vSyncro = ((Robot)robot).Synchro(qi,qf,vqMax);
  APlan = ((Robot)robot).Coef434(tace,tdec,vSyncro);
}
```

Figure 7. ODEs of basic robot motion and Java code to program the 4-3-4 polynomial trajectory

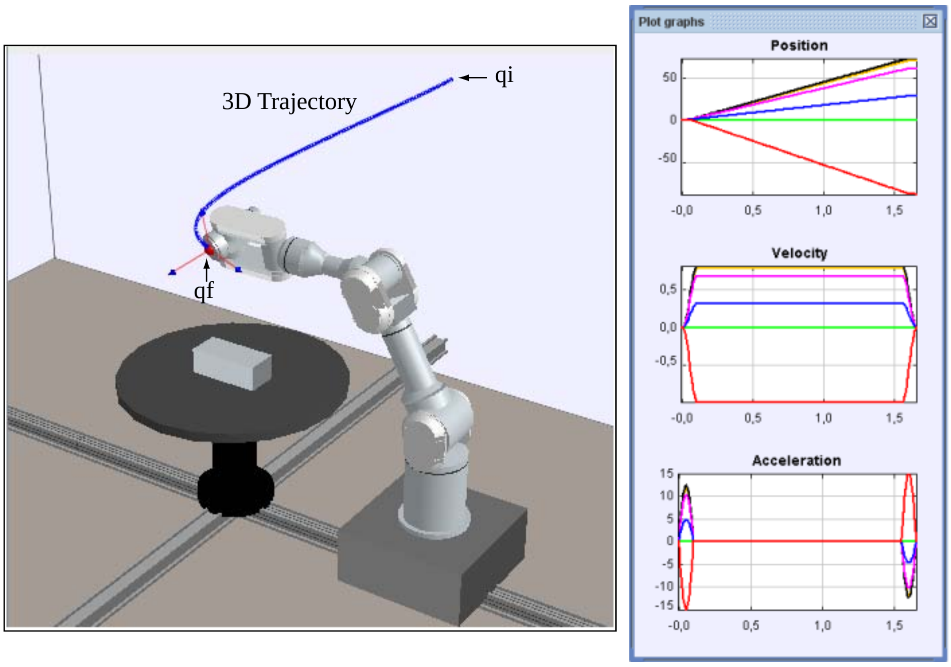
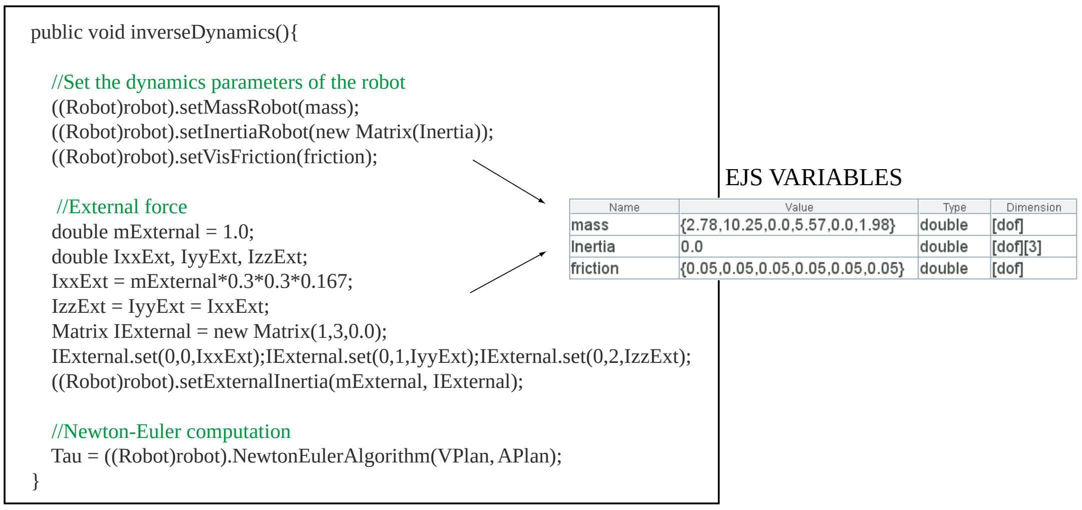


Figure 8. Simulation of a 4-3-4 polynomial trajectory for the 6 rotational DOF robot

3.4 Dynamics features

The Robotics module of EjsRL implements numerical methods to solve the forward and inverse dynamics problems. Fig. 9 shows the implementation of the inverse dynamics problem of the 6 rotational DOF robot 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.



```
public void inverseDynamics(){

   //Set the dynamics parameters of the robot
   ((Robot)robot).setMassRobot(mass);
   ((Robot)robot).setInertiaRobot(new Matrix(Inertia));
   ((Robot)robot).setVisFriction(friction);

    //External force
   double mExternal = 1.0;
   double IxxExt, IyyExt, IzzExt;
   IxxExt = mExternal*0.3*0.3*0.167;
   IzzExt = IyyExt = IxxExt;
   Matrix IExternal = new Matrix(1,3,0.0);
   IExternal.set(0,0,IxxExt);IExternal.set(0,1,IyyExt);IExternal.set(0,2,IzzExt);
   ((Robot)robot).setExternalInertia(mExternal, IExternal);

   //Newton-Euler computation
   Tau = ((Robot)robot).NewtonEulerAlgorithm(VPlan, APlan);
}
```

EJS VARIABLES

Name	Value	Type	Dimension
mass	{2.78,10.25,0.0,5.57,0.0,1.98}	double	[dof]
Inertia	0.0	double	[dof][3]
friction	{0.05,0.05,0.05,0.05,0.05,0.05}	double	[dof]

Figure 9. Programming the inverse dynamics with an external force of the 6 rotational DOF robot

3.5 Adding computer vision features for visual servo control

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). Moreover, EJS+EjsRL provides a functional framework for easily reading and writing images from/to the interface.

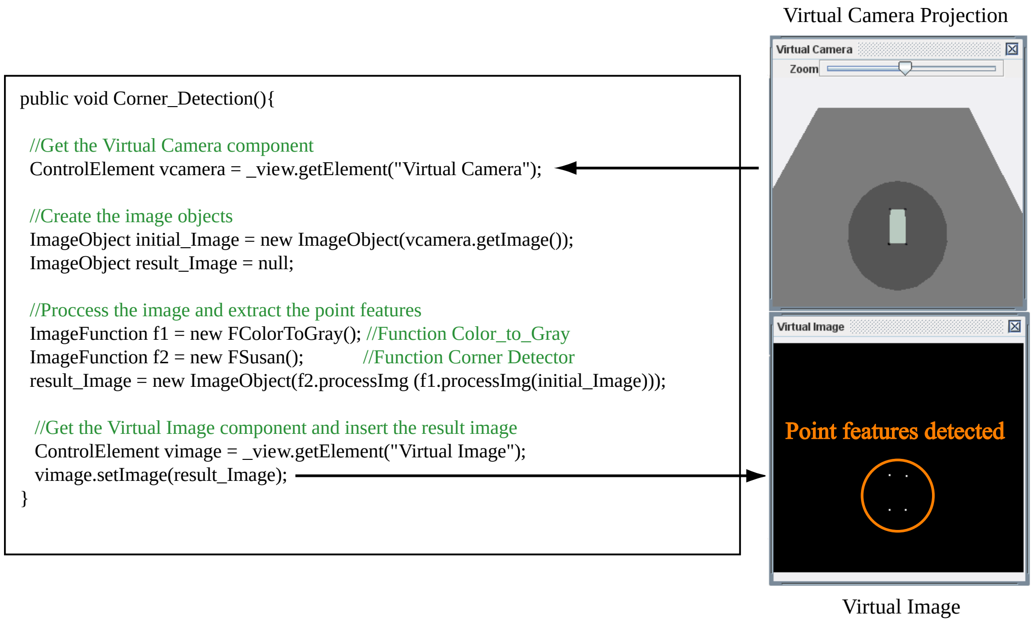
In order to show the capabilities of Computer Vision package for easily implementing a computer vision algorithm in the virtual robotic environment previously created, the proposed simulation deals with an EIH vision based control using four corner features in the control loop. The control action and the interaction matrix used in this control algorithm are based on a classical 2D visual servoing task [31], according to the following expressions:

$$\mathbf{v}_c = -\lambda \, \hat{\mathbf{L}}_s^{+} \left(\mathbf{s} - \mathbf{s}^* \right) \tag{1}$$

$$\hat{\mathbf{L}}_s = \begin{bmatrix} -1/Z & 0 & x/Z & xy & -(1+x^2) & y \\ 0 & -1/Z & y/Z & 1+y^2 & -xy & -x \end{bmatrix} \tag{2}$$

where *s* are the current visual features, *s** are the desired visual features, and λ is the proportional controller; (*x*, *y*) are the point coordinates of each feature; and *Z* is the current distance from the EIH camera to the each feature.

In order to implement the visual servo control, firstly it is necessary to obtain a view projection from the end effector of the robot. For that end, EJS has a property which allows users to create a virtual camera in the 3D robotic environment. Fig. 11 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. Fig. 10 shows the Java code which computes corner detection in the virtual camera’s image. 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) and the point features are detected using one of the implemented algorithms, for example the SUSAN method [32]. These point features can be seen in the window “Virtual Image” of the Figures 10 and 11. The evolution of both velocity module and point features showed in Fig. 11 validates the correct convergence of the visual servo task.



```
public void Corner_Detection(){

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

  //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(initial_Image)));

   //Get the Virtual Image component and insert the result image
   ControlElement vimage = _view.getElement("Virtual Image");
   vimage.setImage(result_Image);
}
```

Figure 10. Java code for the implementation of the point features detection in the EIH virtual camera

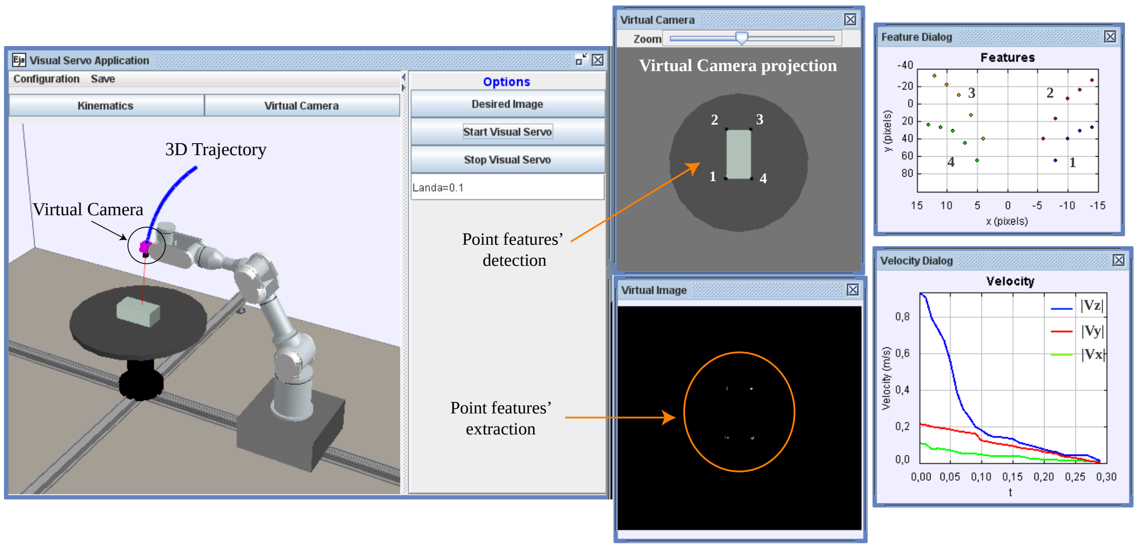


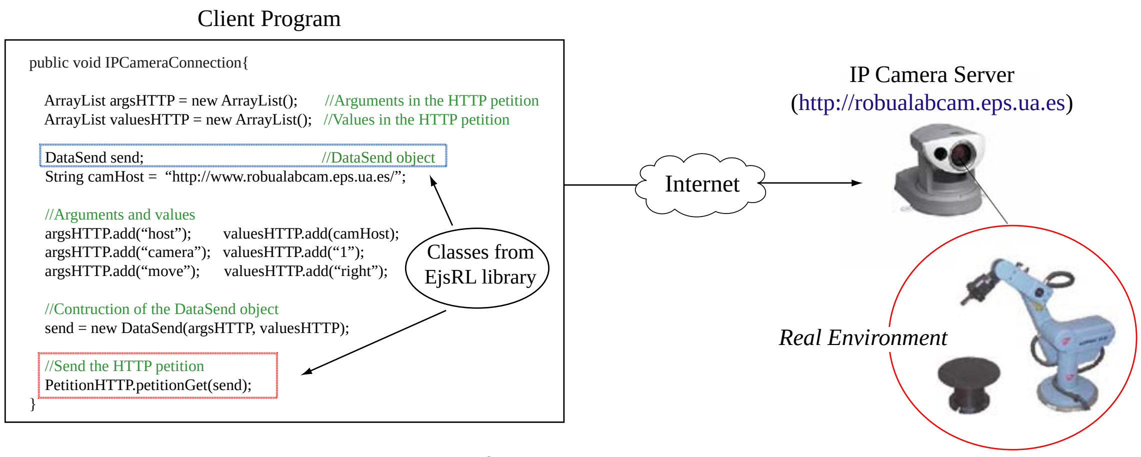
Figure 11. Appearance of the final application for the simulation of a visual-servoing task using point features

4. Advanced features

4.1. Remote operation support

The Internet communication methods implemented in EjsRL are based on GET messages of HTTPS. In addition, with HTPPS, all the data are encrypted, which allow applications to communicate over secure connections.

The Remote Operation classes can be used for several remote operations such as the teleoperation by high-level commands of real devices. The control of an IP camera from a client interface could be an example of that. IP cameras usually have a web server and their pant-tilt-zoom movements can be remotely controlled. Fig. 12 shows an example where a simple Java program moves an IP camera (*camHost* parameter). This client program developed in EJS environment sends a HTTPS request to the camera host. It is worth noting the classes which implement the communication engine are *DataSend* and *PetitionHTTP*. The first one wraps all the user arguments (*argsHTTP* variable) and their corresponding values (*valuesHTTP* variable) in an object for the HTTPS command. Secondly, the class *PetitionHTTP* gets the *DataSend* object, builds the HTTPS request and sends it.



```
public void IPCameraConnection{

  ArrayList argsHTTP = new ArrayList();      //Arguments in the HTTP petition
  ArrayList valuesHTTP = new ArrayList();   //Values in the HTTP petition

  DataSend send;                                       //DataSend object
  String camHost =  "http://www.robualabcam.eps.ua.es/";

  //Arguments and values
  argsHTTP.add("host");        valuesHTTP.add(camHost);
  argsHTTP.add("camera");   valuesHTTP.add("1");
  argsHTTP.add("move");       valuesHTTP.add("right");

  //Contruction of the DataSend object
  send = new DataSend(argsHTTP, valuesHTTP);

  //Send the HTTP petition
  PetitionHTTP.petitionGet(send);
}
```

Figure 12. Remote control of an IP camera using HTTPS communication

4.2. External connection with Matlab

EJS has a connection with Matlab/Simulink which lets users specify and solve their models with the help of these tools [29]. Thus, the model can be defined with Matlab code in a Matlab function (m-file), with a Simulink block diagram or with both. Fig. 13 shows the appearance of an application developed using this feature about a decoupled control of a 3 rotational DOF robot where the electrical model is computed by a Simulink diagram and the path planning together with the 3D graphical simulation (mechanical part) are developed using EJS+EjsRL. In the upper part, it can be seen the simulation of the robot with its respective plot controls, which show the input and the output values. Simulink diagram is set up by the PID control of each DOF, the power amplifier stage and the engine blocks with the model of a DC motor. The torque values are transferred to the forward dynamics method of EjsRL to compute the acceleration for the path planning algorithm. Feedback variables *q* and *v* are values obtained directly from the path planning and connected with the Simulink blocks.

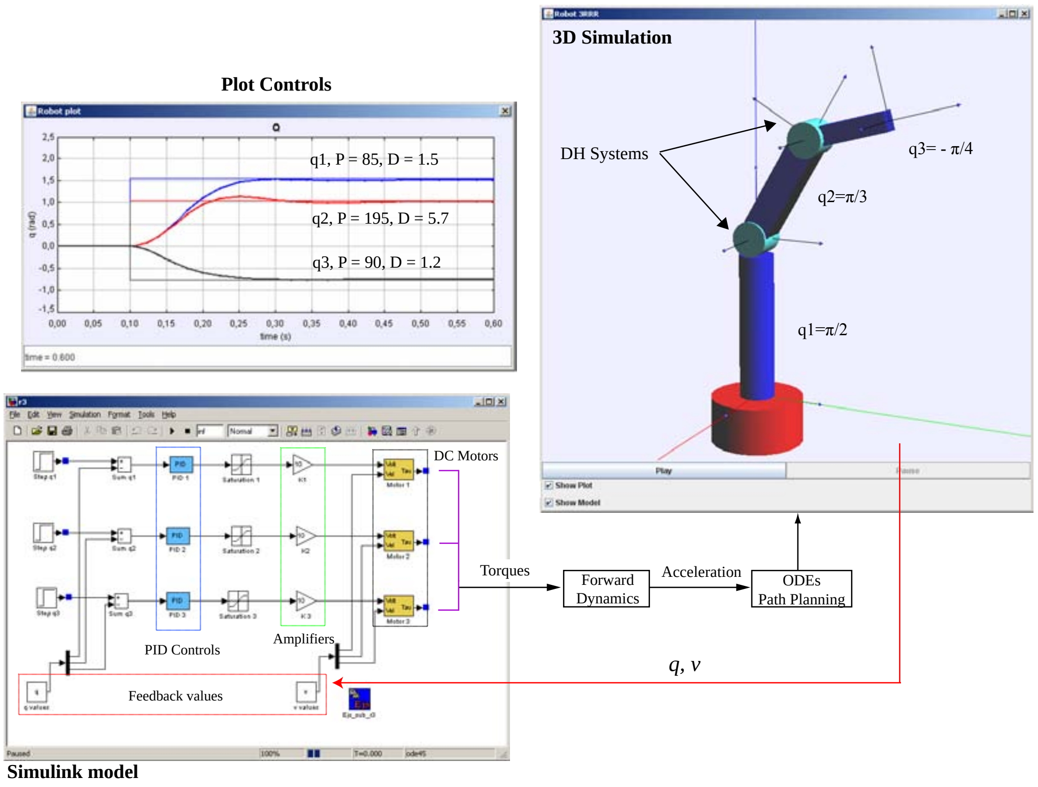


Figure 13. Position control of a 3 rotational DOF using EJS+EjsRL and Simulink

4.3 Augmented reality features

EJS contains the AR panel, a control where virtual worlds can be complemented with real information inside the same 3D environment. The functionality of this control is based on the insertion of an image at the background of the 3D environment and to project the virtual objects with the same position parameters as the world. In order to do that, it is only necessary to specify the direction of the image in the AR panel's properties and the extrinsic parameters of the real camera (position and orientation of the world reference system). Fig. 14 shows an example of an application which uses the AR panel. Here, an image from a real robotic plant is combined with a virtual environment that models some of the objects of the real site.

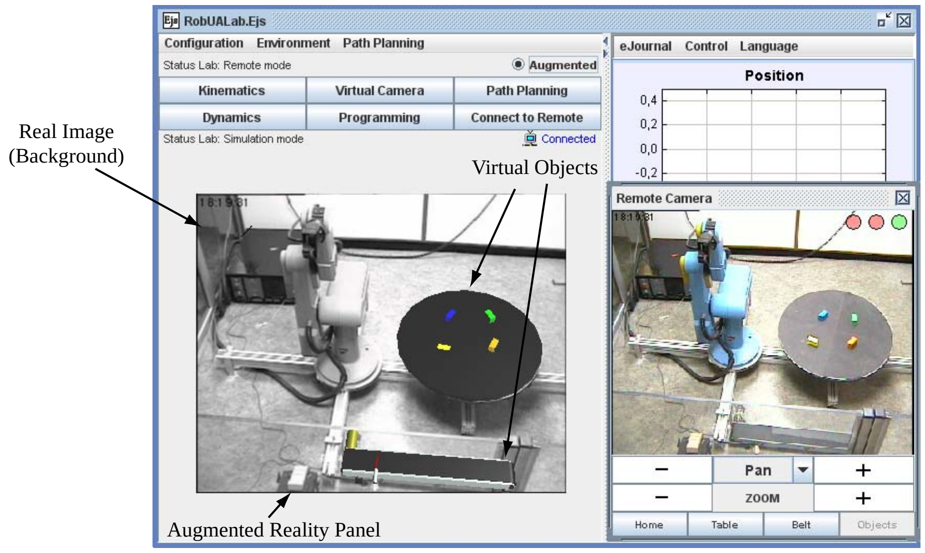


Figure 14. An application which uses the AR panel

4.4. Collision detection

The 3D components of EJS contain certain properties for object interaction. The most important one is the object collision detection. If this property is activated, the 3D object will be able to detect the collision with other 3D objects of the virtual environment that have activated this option. Fig. 4 (see Section 2) shows the properties of the VRML components which compose the end effectors of each one of the robot. The field "Collision" is activated (true) in both objects, so the simulation will detect the collision when the objects collide. User can program the action to do when the 3D object has detected a collision, for example stop the robot trajectory.

5. Other experimental examples developed with EJS+EjsRL

This section describes other advanced examples whose development is based on EJS+EjsRL. Fig. 15 shows the appearance of these applications. These examples and other can be accessed from the following web page dedicated to EJS+EjsRL: http://www.aurova.ua.es/rcv.

The first example (Fig. 15.a) is a virtual and remote laboratory for training and learning in Robotics. All the R&CV features such as kinematics, path planning, programming, object recognition and remote operation has been developed using the library EjsRL. The system, called *RobUALab.ejs* [27, 28], can be accessed from http://robualab.eps.ua.es and allows users to simulate and test positioning commands for a robot by means of a virtual environment, as well as execute high level commands in a real remote robot through HTTPS commands. In addition, it has an interface with augmented reality support where images of the real robot are complemented with some data from the virtual environment. Currently, this application belongs to a network performed by different virtual and remote laboratories from Spanish universities, called "AutomatL@bs" (http://lab.dia.uned.es/automatlab/index_en.html).

The other two experimental applications are about multi-robotic systems. EjsRL allows users the instantiation of different robot objects since it has been created in an object-oriented form. The second application (Fig. 15 b) is a robotic hand simulated which represents virtually a real system developed by the company Barrett Technology [32]. Fig. 15b shows the interface of the application developed with EJS+EjsRL, where it can be seen the 3D virtual environment. In this simulation has been included some robotic concepts related with kinematics, path planning and dynamics. The third simulation (Fig. 15 c) is about a multi-robotic system composed by two manipulators: a PA-10 robot of 7 rotational DOF and a 3 rotational DOF robot (RRR). This last serial robot is coupled to a link of the upper part of the PA-10. Furthermore, the robot RRR has a virtual camera at the end as an EIH configuration. Fig. 15c shows the interface of the application developed. On the left, it can be seen the 3D virtual environment, which displays the workspace where the robots are located. On the right, it is represented the virtual projection of the EIH camera located at RRR.

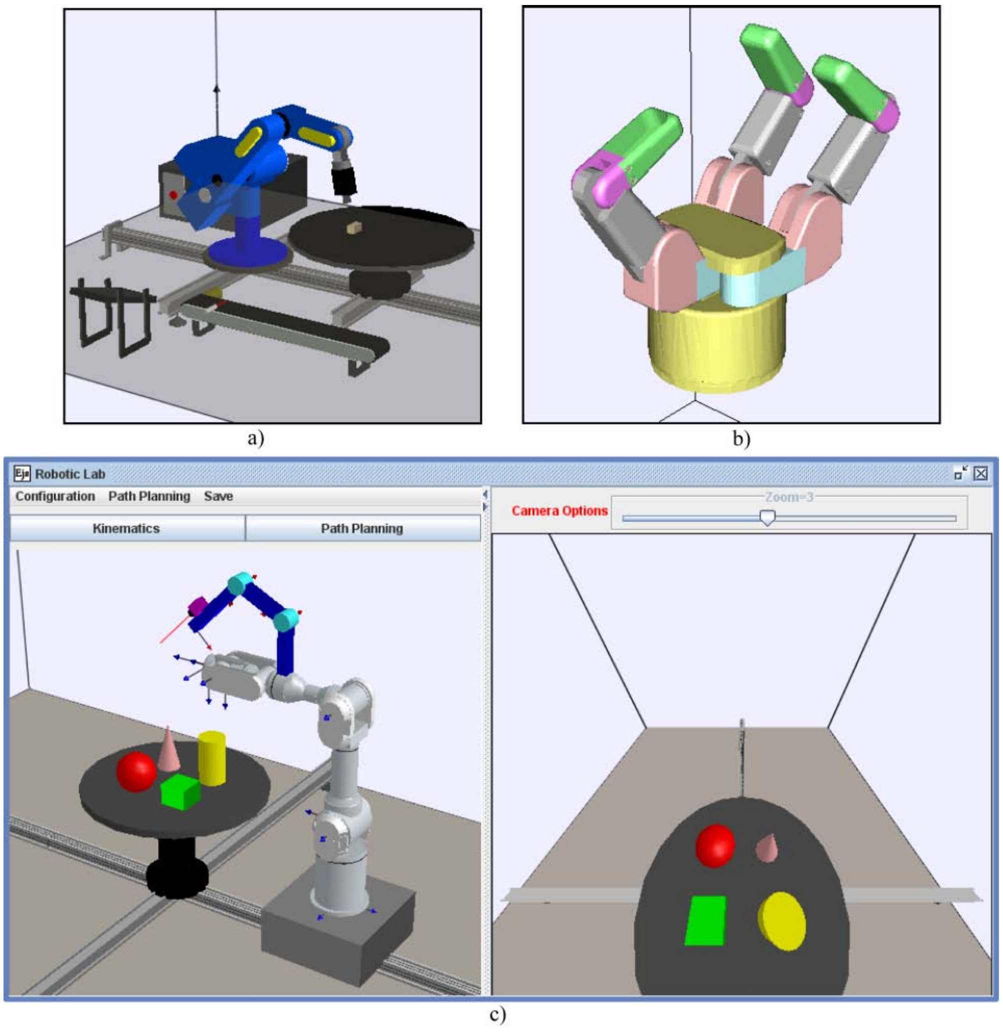


Figure 15. Several advanced robotic applications developed using EJS+EjsRL

6. Computational efficiency.

Performance and accuracy are two very important aspects to be considered for any modeling and simulation software tool. This section deals with the comparison between EJS+EjsRL and other software tools in relation with these capabilities.

The majority of the software tools for R&CV are based on three programming languages: C/C++, Java and Matlab. The performance and accuracy of the software tools are directly related with the computational efficiency of the programming language and with the algorithm's accuracy. Thus, in order to perform a good comparison, different tools related with these programming languages were chosen.

The comparison is based on three test cases, one for Robotics, another for Computer Vision and finally another for R&CV. In the case of the Robotics test, a comparison between the computation accuracy of the inverse dynamics was made for the 6 rotational DOF robot previously considered (Section 3) with the mass and inertial properties showed in Fig. 9. With regard to the Computer Vision test, the computation time of the Harris algorithm on the same image was measured. Finally, the test about R&CV is the computation time of an iteration of the virtual visual servo control showed in Section 6.

The hardware platform used for the comparison tests was a PC Intel Pentium running at 1.73 GHz with 1GB of RAM. The Robotics test were computed for a zero pose, with null joint velocities and accelerations, no external forces acting on the end-effector, and a gravity force along the waist axle of the manipulator. The Computer Vision test was made using the image showed on Fig. 10 (256x256 pixels, RGB 8bpp). The computation time measured includes loading the initial image, processing the algorithm and displaying the resulting image. The results obtained in both cases are reported in Table 2. Finally, the R&CV test was based on the example show in Section 3. The computation time ranges from the feature extraction of the image to the velocity computation of the robot. The comparison performed in shown in Table 3.

	Robotics Test (Nm)			**Computer Vision Test (ms)**		
Language	Matlab	C/C++	Java	Matlab	C/C++	Java
Tool	Robotics Toolbox	RoboOp	EJS+EjsRL	CV Toolbox	OpenCV	EJS+EjsRL
Results	{0, 0, -87.0392, 0, 0, 0}	{0, 0, -87.0392, 0, 0, 0}	{0, 0, -87.0392, 0, 0, 0}	125	60	85

Table 2. Comparison of performance and accuracy between EJS+EjsRL and other tools I

	R&CV Test (ms)		
Language	Matlab	C/C++	Java
Tool	Robotics Toolbox	ViSP	EJS+EjsRL
Results	95	25	75

Table 3. Comparison of performance and accuracy between EJS+EjsRL and other tools II

With regard to the results obtained in the Robotics test, the approach presented here is in a level of accuracy very close to Matlab and C++, which are usually considered to be very precise for a great number of applications. In the case of the Computer Vision and R&CV tests, it is clear that OpenCV and ViSP have a higher computational efficiency that other since are based on C++ language. However, EJS+EjsRL (Java) is faster than Robotics and CV Toolbox of Matlab. In addition, in these computer vision operations only 15 ms is used to load and show the image. In this way, the computational efficiency of EJS+EjsRL regarding interactive capabilities is very high, because it utilizes a minimum time to process user interactions.

7. Conclusions

In this paper, a free Java-based software platform for the creation of advanced robotic and computer vision applications has been presented. This new tool is composed by two parts: 1) a high-level Java library called EjsRL, which provides a complete functional framework for modeling and simulation of robotic and computer vision systems; 2) EJS, an open-source tool which provides full graphical interface support. The integration of both parts permits to create complex and advanced robotic applications, since provides a higher number of functionalities than other software platforms available nowadays.

EjsRL allows users to model complex robotic systems within EJS, to perform high-level Computer Vision algorithms and to execute remote operations. All the classes have been implemented in the same platform, a feature which has not been developed before in other tools. For modeling arbitrary serial-link manipulators, users only have to specify their DH parameters and their physical properties. Then, the Java platform creates an object which has embedded all the robot behaviour (kinematics, dynamics, programming, etc.). In the case of computer vision methods, the library contains a lot of implemented algorithms (Computer Vision classes), in addition to input/output operations for easily reading and writing images. Finally, the Remote Operation functions provide client-server communication support over the HTTPS protocol which can be used to develop applications with remote operation features.

The user-friendly interface of EJS enables users to easily and quickly create advanced Java applications. In addition to its full computer graphical support, this software provides several advanced features such as VRML and OBJ extern file importation and object collision detection. Thus, the approach presented is very suitable to develop research and educational applications in the R&CV fields apart from adding novel features that are not found in other toolboxes available today.

Finally, the paper presents several high-level applications, which combine both R&CV functionalities, in order to validate some of the system capabilities. These experimental examples illustrate a part of the possibilities of the tool presented. Most of these applications have been used as teaching tools for undergraduate students in several courses about Robotics since 2009. These applications have been positively accepted by the students because they are very grateful to experiments with realistic and interactive virtual environments.

Currently, the library EjsRL is being improved in order to incorporate an interface for the OpenCV library. For that end, there are modules of Java such as JRMI (Java Remote Method Invocation) and JNI (Java Native Interface) which permit to access to other external libraries written in other programming languages.

Obtaining EjsRL and EJS

EjsRL can be obtained from http://www.aurova.ua.es/rcv. In this web page there is a lot of information about how to use the library and readers can execute all the examples showed in the paper. In addition, there is a HTML document that explains with detail all the Java classes. EJS can be downloaded from http://fem.um.es/EjsWiki, where readers can experiment with a lot of simulations developed with this software. In order to use them in any operative system, it is necessary to install two Java runtimes: JRE (Java Runtime Environment) 1.6 and Java 3D 1.5 or higher.

References

[1] RoboWorks Software, Newtonium Web site [Online]. Available: http://www.newtonium.com/.

[2] RoboAssist, Robot modeling and control package, New River Kinematics Inc. [Online]. Available: http://www.kinematics.com/products/educational/robotassist/index.html.

[3] Easy-ROB Software, Easy-ROB Web site [Online]. Available: http://www.easy-rob.com/.

[4] W. E. Honey, M. Jamshidi, ROBO_SIM: A robotics simulation environment on personal computers, Robotics and Autonomous Systems 9 (1992) 305-317.

[5] R. Pelossof, A. Miller, P. Allen, and T. Jebara, An SVM learning approach to robotic grasping, in: Proceedings of the IEEE International Conference on Robotics and Automation, Vol. 21, Taiwan, 2004, pp. 3215-3218.

[6] A. Jaramillo-Botero, A. Matta-Gomez, J. F. Correa-Caicedo, and W. Perea-Castro, ROBOMOSP, IEEE Robotics & Automation Magazine 13 (2006) 62-73.

[7] J Jackson, Microsoft robotics studio: A technical introduction, IEEE Robotics & Automation Magazine 14 (2007) 82-87.

[8] R. Gourdeau, Object-oriented programming for robotic manipulator simulation, IEEE Robotics & Automation Magazine 4 (1997) 21-29.

[9] C. Vibet, Symbolic modeling of robots kinematics and dynamics, Robotics and Autonomous Systems 14 (1995) 301-314.

[10] Modelica Association Website. [Online]. Available: http://www.modelica.org.

[11] F. Marchand, F. Spindler, F. Chaumette, ViSP for visual servoing: a generic software platform with a wide class of robot control skills, IEEE Robotics and Automation Magazine 12 (2005) 40-52.

[12] R. J. Babuska, Matlab Design Environment for Robotic Manipulators, in: Proceedings of the 16th IFAC World Congress, Vol. 16, Prague, 2005.
[13] R. Falconi, C. Melchiorri, Roboticad: An Educational Tool for Robotics, in: Proceedings of the 17th IFAC World Congress, Seoul, 2008, Vol. 17.
[14] P.I. Corke, A Robotics Toolbox for MATLAB, IEEE Robotics and Automation Magazine, 3 (1996) 24-32.
[15] Open source computer vision library. [Online]. Available. http://www.intel.com/research/mrl/research/opencv.
[16] The VIGRA library website. [Online]. Available: http://kogs-www.informatik.uni-hamburg.de/~koethe/vigra.
[17] The Vision-something-Libraries [Online]. Available: http://vxl.sourceforge.net/.
[18] Java imaging and graphics library. [Online]. Available: http://rivit.cs.byu.edu/jigl.
[19] The Java Advanced Imaging website. [Online]. Available: http://java.sun.com/products/javamedia/jai.
[20] The Java Sun website. [Online]. Available: http://java.sun.com.
[21] F. Esquembre, Easy Java Simulations: A software tool to create scientific simulations in Java, Computer Physics Communications 156 (2004) 199–204.
[22] Ejs website. [Online]. Available: http://fem.um.es/EjsWiki/index.php/Main/HomePage.
[23] H. Vargas, J. Sanchez, G. Farias, S. Dormido, R. Dormido, S. Canto, and F. Esquembre, Web-based learning resources for automation technician's vocational training: Illustrated with a heat-flow and liquid level laboratories, in: Proceedings of the IFAC Symposium on Advances Control Education, Vol. 7, Spain, 2006, pp. 5-11.
[24] N. Duro, R. Dormido, H. Vargas, S. Dormido, J. Sanchez, G. Farias et al. An integrated virtual and remote control lab: The three-tank system as a case study, Computing in Science & Engineering, 10 (2008) 50–59.
[25] R. Dormido, H. Vargas, N. Duro, J. Sanchez, S. Dormido, G. Farias, et al. Development of a web-based control laboratory for automation technicians: The three-tank system, IEEE Transactions on Education, 51 (2008) 35–44.
[26] C. Jara, F. Candelas, F. Torres, S. Dormido, F. Esquembre, and O. Reinoso, Real-time collaboration of virtual laboratories through the Internet, Computers & Education, 52 (2009) 126-140.
[27] C. Jara, F. Candelas, F. Torres, An advanced interactive interface for Robotics e-learning. International Journal of On-line Engineering, 4 (2008) 17-25.
[28] C. Jara, F. Candelas, S. Puente, J. Pomares, F. Torres, Practical experiences using RobUALab.ejs: a virtual and remote laboratory for Robotics e-learning, in: Proceedings of 8th IFAC Symposium on Advances in Control Education, Vol. 1, Japan, 2009, pp. 7-13.
[29] J. Sanchez, F. Esquembre, C. Martin, S. Dormido, S. Dormido-Canto, R. Canto, et al. Easy Java Simulations: an open-source tool to develop interactive virtual laboratories using MATLAB/Simulink, International Journal of Engineering Education 21 (2005) 789–813.
[30] J. Denavit and R.S. Hartenberg, A kinematic notion for lower-pair mechanisms based on matrices, Journal of Applied Mechanics, 22 (1995) 215–221.
[31] F. Chaumette and S. Hutchinson, Visual Servo Control, Part I: Basic Approaches, IEEE Robotics and Automation Magazine, 13 (2006) 82-90.
[31] S.M. Smith and J.M. Brady, SUSAN: a new approach to low level image processing, International Journal of Computer Vision, 23 (1997) 45-78.
[32] Barrett Technology, (2009) Barrett Hand. [Online]. Available: http://www.barrett.com/.

Figure Captions

Figure 1. Software architecture of the tool and package-class diagram of EjsRL.

Figure 2. Overview of the EJS' environment.

Figure 3. Java code for the model in order to create a 6 rotational DOF robot.

Figure 4. Interface construction of the Robotics application proposed.

Figure 5. Java code for the forward kinematics of the 6 rotational DOF robot modelled with VRML objects.

Figure 6. Java code for the inverse kinematics implementation.

Figure 7. ODEs of basic robot motion and Java code to program the 4-3-4 polynomial trajectory.

Figure 8. Simulation of a 4-3-4 polynomial trajectory for the 6 rotational DOF robot.

Figure 9. Programming the inverse dynamics with an external force of the 6 rotational DOF robot.

Figure 10. Java code for the implementation of the point features detection in the EIH virtual camera.

Figure 11. Appearance of the final application for the simulation of a visual-servoing task using point features.

Figure 12. Remote control of an IP camera using HTTPS communication.

Figure 13. Position control of a 3 rotational DOF using EJS+EjsRL and Simulink.

Figure 14. An application which uses the AR panel.

Figure 15. Several advanced robotic applications developed using EJS+EjsRL.