A Modeling and Simulation Platform for Robot Kinematics aiming Visual Servo Control

Lélio R. Soares Jr. and Victor H. Casanova Alcalde
Electrical Engineering Department, University of Brasilia
Brazil

1. Introduction

A robotic system is a mechanical structure built from rigid links connected by flexible joints. The arrangement of links and joints (robot architecture) depends on the task the robot was designed to perform. The robot links have then different shapes and the joints can be of revolute (rotational motion) or prismatic (translation motion) nature. These robots, as described, perform task on an open-loop control scheme, i.e. there is not feedback from the environment (robot workspace) thus it will not notice changes in the workspace. As an attempt to establish a closed-loop control scheme a computer-based vision systems is introduced to detect workspace changes and also to allow guiding the robot (Hutchinson et al., 1996).
At the University of Brasilia to cope with the study and teaching of robotics an educational robotic workstation was built around the Rhino XR4 robot (Soares & Casanova Alcalde, 2006). To implement a vision-guided robot a video camera was installed and integrated to the robot control system. As an alternative for dealing with the real system and for teaching purposes a simulation platform was devised within the Matlab environment (Soares & Casanova Alcalde, 2006). The platform was called *RobSim* and it is based on assembling elementary units (primitives) which represent the robot links, being the joints represented by the motion they perform. This simulation and developing platform then evolved and now it includes robot visual servo control being presented in this work. Within *RobSim* platform control algorithms can be developed for the vision-guided robot to perform tasks before implementing them on the real system.
Simulation tools for either conventional robotic systems (Legnani, 2005; Corke, 1996) and for vision-based systems (Cervera, 2003) do exist, this work presents a unified environment for both systems. The developed simulation tools were assembled as a laboratory platform, where robotic and vision-based algorithms share similar data structures and block building methodologies. Moreover, this platform was developed mainly for educational purposes; later on it was found it can be used for research and design of robotic systems. The graphical presentation is as simple as possible, but allowing an insight and visualization of parts and motions.
The chapter is organized as follows; initially the *RobSim* basic mounting blocks, the primitives, are defined and described. Then, the *RobSim* developed Matlab functions for initialization, motion, computer display and image acquisition are presented. Following, the modeling and simulation capabilities *RobSim* platform offers are presented together with

Source: Visual Servoing, Book edited by: Rong-Fong Fung,
ISBN 978-953-307-095-7, pp. 234, April 2010, INTECH, Croatia, downloaded from SCIYO.COM

applications to fixed and mobile robots. Further on, vision-based control schemes are briefly discussed. Finally, implementation of visual-based control schemes applied to a robotic workstation consisting of a Rhino XR4 robot and a computer vision system is considered. Image- and position-based visual servoing schemes are implemented.

2. *RobSim* – a modeling and simulation platform for robotic systems

In order to model and simulate the kinematics of robotic systems a software platform named *RobSim* was developed. Three types of basic elements were defined to assembly a model for a vision-guided robotic system: block, wheel and camera. Being basic elements they will be called *primitives*. They will be sufficient to assembly a simulation model for robotic manipulators and robotic vehicles guided by a computer vision system.

2.1 Block primitive

The block primitive is defined as a regular polyhedron with rectangular faces. The faces meet along an edge and three of these intersect orthogonally at a vertex. A block primitive consists then of six faces, twelve edges and eight vertexes. Figure 1 shows a block primitive with its allocated coordinate frame $\{X_b,Y_b,Z_b\}$. The frame orientation is assigned as follows, the X_b-axis along the block length (L), the Y_b-axis along the block width (W) and the Z_b-axis along the block height (H). A general graphical reference coordinate frame $\{X_g,Y_g,Z_g\}$ is also shown in Figure 1, it indicates the block viewing angle for displaying purposes.

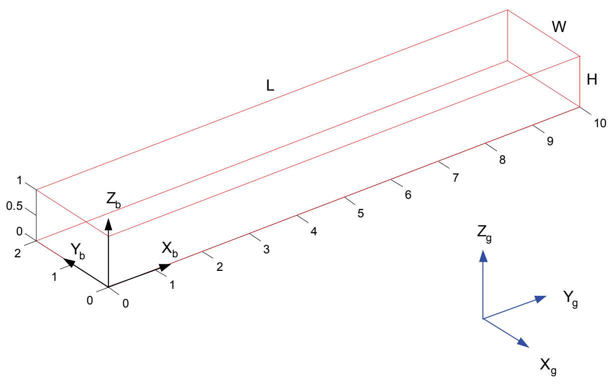

Fig. 1. A Block Primitive

A block primitive will be geometrically defined by nine components: a) eight vectors, each one corresponding to the *3D* coordinates of its vertexes; and b) a character identifying the assigned color to the line edges.

2.2 Wheel primitive

For simulating wheeled mobile robots a wheel primitive is defined. The wheel primitive is defined as two circles of equal radius assembled parallel to each other at a certain distance. The wheel rotation axis passes through the centers of both circles. Figure 2 shows a wheel primitive with its allocated coordinate frame. The wheel primitive coordinate frame

$\{X_b, Y_b, Z_b\}$ is attached to the wheel primitive, being its origin fixed at the middle of the internal line between the circle centers. The Z_b-axis coordinate is fixed along the rotation axis, the X_b-axis along the initial rotation angle (0º).

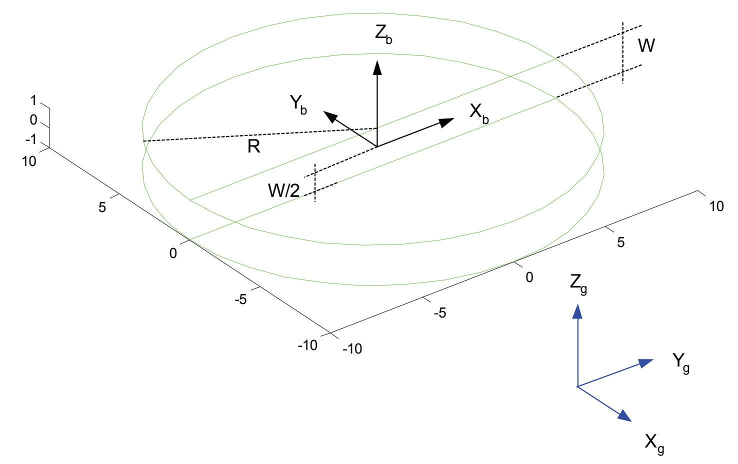

Fig. 2. A Wheel Primitive

A wheel primitive will be geometrically defined by four components: a) the circle radius (R); b) the distance between the circle centers (W); c) the number of points defining both circumferences; and d) the color identifying character.

2.3 Camera primitive

For vision-guided robotic systems, manipulators or mobile robots, video cameras are required. Then, a camera primitive was developed from a modified block primitive. It is a small regular polyhedron with rectangular faces but having a larger opening on one extreme representing the light capturing entrance. Figure 3 shows a camera primitive with its coordinate frame $\{X_b, Y_b, Z_b\}$. The camera primitive coordinate frame is attached to the opposite face, where the image is formed. The coordinate frame center is fixed at the center of this rectangular face, the Z_b-axis along the camera length (L), the X_b-axis along the camera height (H) and the Y_b-axis along the camera width (W). This orientation follows the computer vision convention, so the Z_b-axis coincides with the camera optical axis.
Due to its particular function, a camera primitive will be defined by three groups of components: a) twelve vectors to characterize its vertexes spatial coordinates; b) a color identifying character; and c) the camera intrinsic parameters (subsection 3.4).

3. *RobSim* processing functions

Within the Matlab environment *RobSim* functions for processing the primitives were developed. These functions allow: defining the primitives (initialization functions); moving the primitives (moving functions); and displaying the primitives (displaying functions). An image acquisition function to simulate image capture was also developed.

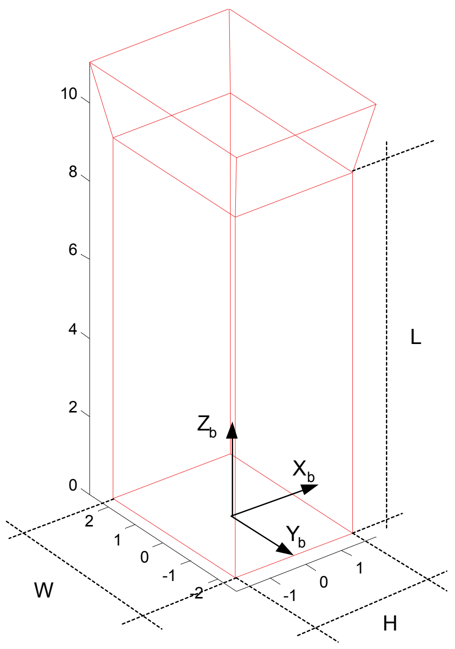

Fig. 3. A Camera Primitive

3.1 Primitives initialization functions

The primitives have to be introduced to the Matlab environment. For that, Matlab structure-type variables (*struct*) are used for initialization of the primitives being the dimensions expressed in centimeters.

Initializing a block primitive – The function to initialize the block primitive *struct* variable has the following syntax:

- *blk=init_block(L,W,H,color)*
  where *L*, *W*, *H* and *color* are respectively the length, width, height and line color of the block primitive.

Initializing a wheel primitive – The function to initialize the wheel primitive *struct* variable is

- *circ=init_circ(R,W,n,color)*
  where *R*, *W*, *n* and *color* are respectively the radius, width, number of circumference points and line color of the wheel primitive.

Initializing a camera primitive – The function to initialize the camera primitive *struct* variable is

- *cam= init_cam(L,W,H, f,px,py,alpha,u0,v0,color)*
  where *L*,W,*H*, and *color* are respectively the length, width, height and line color of the camera primitive. The parameters *f*, *px*, *py*, *alpha*, *u0* and *v0* are the camera intrinsic parameters (Chaumette & Hutchinson, 2006). These camera intrinsic parameters will be further discussed in subsection 3.4.

3.2 Primitives moving functions

Once defined the primitives within Matlab, other functions are necessary for moving the primitives as they simulate the different moving robotic links. For moving the primitives all of its characteristic points have to be moved. A homogeneous transformation (Schilling, 1990) is then applied upon the vectors which define those characteristic points.

Moving a block primitive - Given a struct variable *blk_i* representing an initial block primitive pose (position and orientation), a new variable *blk_o* will represent the final primitive pose as a result of a moving function. For a block primitive the developed moving function is

- *blk_o=move_block(blk_i,***R***,***t***)*
  where **R** and **t** are respectively the rotation matrix (3×3) and the translation vector (3×1) of the homogeneous transformation representing the executed motion.

Moving a wheel primitive - Given *circ_i* representing an initial wheel pose, after applying a moving function the wheel primitive will assume a final pose *circ_o*. For this action the developed moving function is

- *circ_o=move_circ(circ_i,***R***,***t***)*
  where **R** and **t** have the same meaning as the block primitive.

Moving a camera primitive – Similarly, for an initial pose of the camera primitive *cam_i*, a final pose *cam_o* is achieved after a moving function. A developed camera primitive moving function is

- *cam_o=move_cam(cam_i,***R***,***t***)*
  where **R** and **t** have the same meaning as for the block and wheel primitives motion.

3.3 Primitives displaying functions

For displaying primitives specific functions were developed around the Matlab built-in *plot3* function. As the vertexes define the geometry of primitives, for displaying purposes straight lines were drawn to join the vertexes. Thus the displayed primitives look like a *wire-frame* model for solid objects. The graphic displaying functions developed for primitives are

- *plot_block(block)*
- *plot_circ(circ)*
- *plot_cam(cam)*

for the block, wheel and camera primitives respectively. The function argument in the three cases is precisely the *struct* variable that represents the primitive.

3.4 Image acquisition function

A computer-based vision system for robotic systems demands video cameras. A camera coordinate frame is attached to the camera, being ${}^{0}\mathbf{T}_c$ the homogeneous transformation matrix relating the camera position (**t**) and orientation (**R**) referred to the base coordinate frame. **R** and **t** constitute the camera extrinsic parameters which together with the intrinsic parameters {*f, px, py, α, u0, v0*} are used to setting up the camera primitive. These intrinsic parameters arise from the perspective projection model (Hutchinson et al., 1996) adopted for the camera and are shown graphically in Figure 4.

An image acquisition function *point_view* was developed to simulate an image point capture and its syntax is

- *pimag=point_view*($\mathbf{p}_{3D}$,**K***i*,${}^{0}\mathbf{T}_c$)
  where $\mathbf{p}_{3D}$ is a vector representing the *3D* position of a point in the camera *field-of-view* (FOV), relative to base frame; **K***i* is the matrix of the camera intrinsic parameters; and

pimag will return the $\mathbf{p}_{imag}$, the 2D position of the image point measured in *pixels*. **K***i* is arranged as follows

$$\mathbf{K}_i = \begin{bmatrix} -\dfrac{f}{p_x.\cos\alpha} & 0 & u_0 \\ \dfrac{f.\tan\alpha}{p_y} & \dfrac{f}{p_y} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

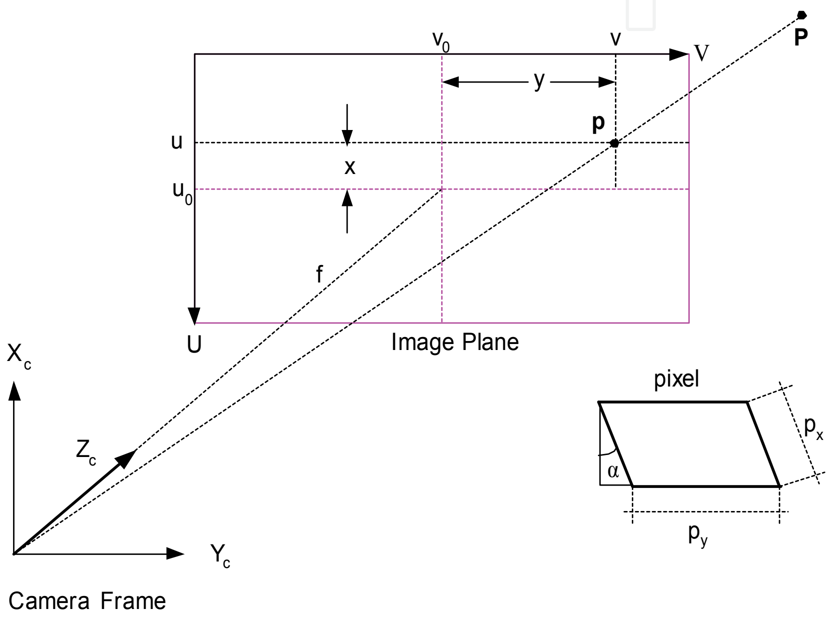

Fig. 4. Perspective Projection Model for the Camera

4. Modeling and simulation of robotic systems kinematics using *RobSim*

A robotic manipulator or vehicle can be considered as a chain of rigid links interconnected by either revolute or prismatic joints. The proposed modeling and simulation tool *RobSim* associates a primitive to a robotic link. By programming the primitive initialization, moving and displaying functions together with Matlab built-in functions it is possible to simulate the kinematical model of any robotic structure. Thus, from these basic structures, the primitives, the kinematics of complex robotic systems can be simulated for analysis and design purposes.
Within *RobSim* the robot joints are not graphically represented or displayed, being their nature (prismatic- or revolution-type) revealed as the motion progresses. For this reason, different colors must be assigned for primitives representing consecutive links.
As primitives are represented by a structure-type variable, the whole set of assembled primitives representing the robot system will be a higher-level structure-type variable.
The kinematical model of a robotic system is determined by applying the Denavit-Hartenberg (DH) algorithm (Schilling, 1990). Transformations between successive links *(k-1)* and (*k*) are characterized by homogenous transformation matrixes like

$$^{(k-1)}\mathbf{T}_k = \begin{bmatrix} \mathbf{R}_{3\times3} & \mathbf{t}_{3\times1} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} \tag{2}$$

In which $\mathbf{R}_{3\times3}$ is the rotation matrix representing the relative orientation between frames and $\mathbf{t}_{3\times1}$ is the translation vector representing the relative position between the frames origins.
By using DH kinematical parameters $\{\theta, d, a, \alpha\}$, Equation (2) can be written as

$$^{(k-1)}\mathbf{T}_k = \begin{bmatrix} C\theta_k & -C\alpha_k S\theta_k & S\alpha_k S\theta_k & a_k C\theta_k \\ S\theta_k & C\alpha_k C\theta_k & -S\alpha_k C\theta_k & a_k S\theta_k \\ 0 & S\alpha_k & C\alpha_k & d_k \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

In which for rotational joints, θ is the joint variable and C and S represent the cosine and sine functions respectively. To illustrate DH modeling and link-primitive assignment correspondences, Figure 5 shows the coordinate frame assignment for two robotic links. For these links, Figure 6 shows the assembling of primitives
The kinematical model of a particular robot of n joints will be the homogeneous transformation relating the tool-tip coordinate frame (frame n) to the base coordinate frame (frame 0) obtained as

$$^{0}\mathbf{T}_n = {}^{0}\mathbf{T}_1 \cdot {}^{1}\mathbf{T}_2 \cdots {}^{k-1}\mathbf{T}_k \cdots {}^{n-1}\mathbf{T}_n \tag{4}$$

An additional transformation will be necessary for displaying purposes relating the base coordinate frame to the displaying frame $^{g}\mathbf{T}_0$ (Fig. 6).

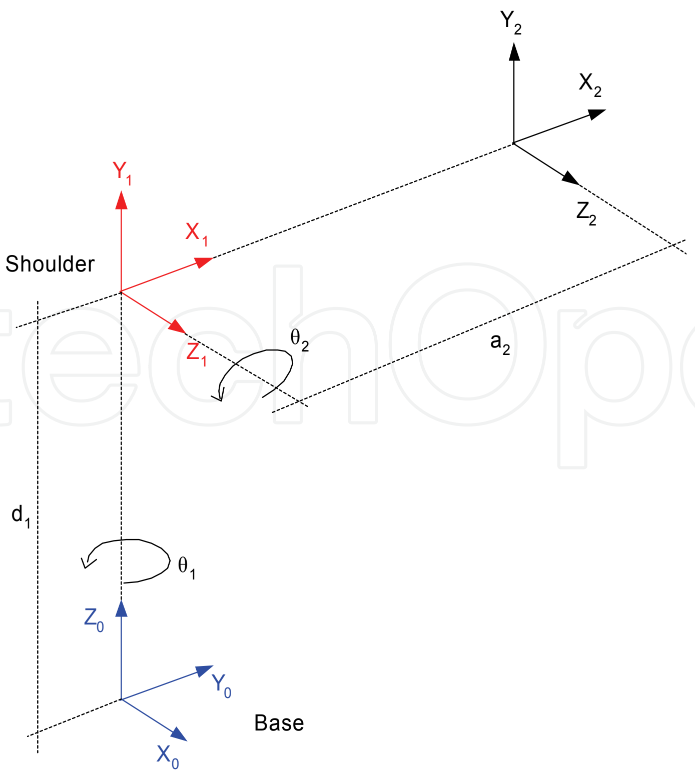

Fig. 5. DH Link Coordinates for two robotic links

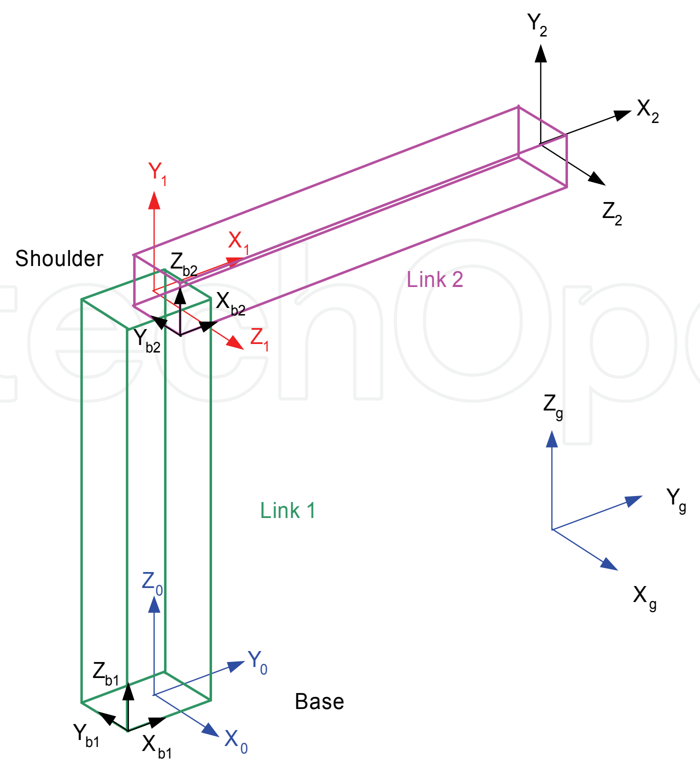

Fig. 6. Assembling Primitives for two robotic links

4.1 *RobSim* modeling and simulation procedure

The different stages to assembly a *RobSim* simulation model for a given vision-guided robotic system are:

1. Allocating link coordinates and determining the kinematical parameters for the robotic system according to the Denavit-Hartenberg (DH) algorithm;
2. Representing the different robot links by the block, wheel or camera primitives as applied;
3. Assembling the chosen primitives through their coordinates as referred to the link coordinates determined by the DH algorithm;
4. Determining the primitives configuration referred to the robot base coordinates;
5. Developing the robotic system initialization as a Matlab *struct* variable, whose variable fields are the individual primitives *struct* representations;
6. Developing the moving and displaying functions for the robotic system from the individual primitives moving and displaying functions;
7. Generating trajectories and executing tasks by controlling the joint variables of the simulation model.

4.2 Simulation of robotic systems

Initially a *RobSim* model for the Rhino XR4 robot will be developed and a simulation test executed. The Rhino XR4, shown in Fig. 7, is an educational desktop robot, classified as a five-axis electric-drive articulated coordinates robot. Around this robot an educational robotic workstation (Soares & Casanova Alcalde, 2006) was built.

Applying the *RobSim* modeling and simulation procedure, link coordinates were allocated and the kinematical parameters for the Rhino XR4 robot obtained, as shown in Figure 8.

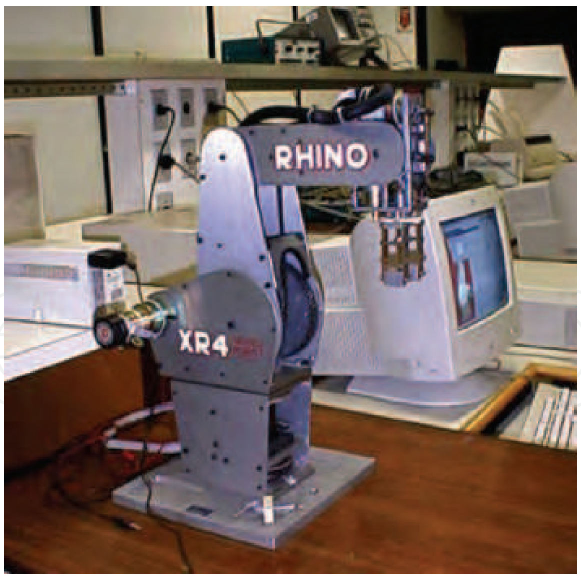

Fig. 7. The Rhino XR4 Educational Robot

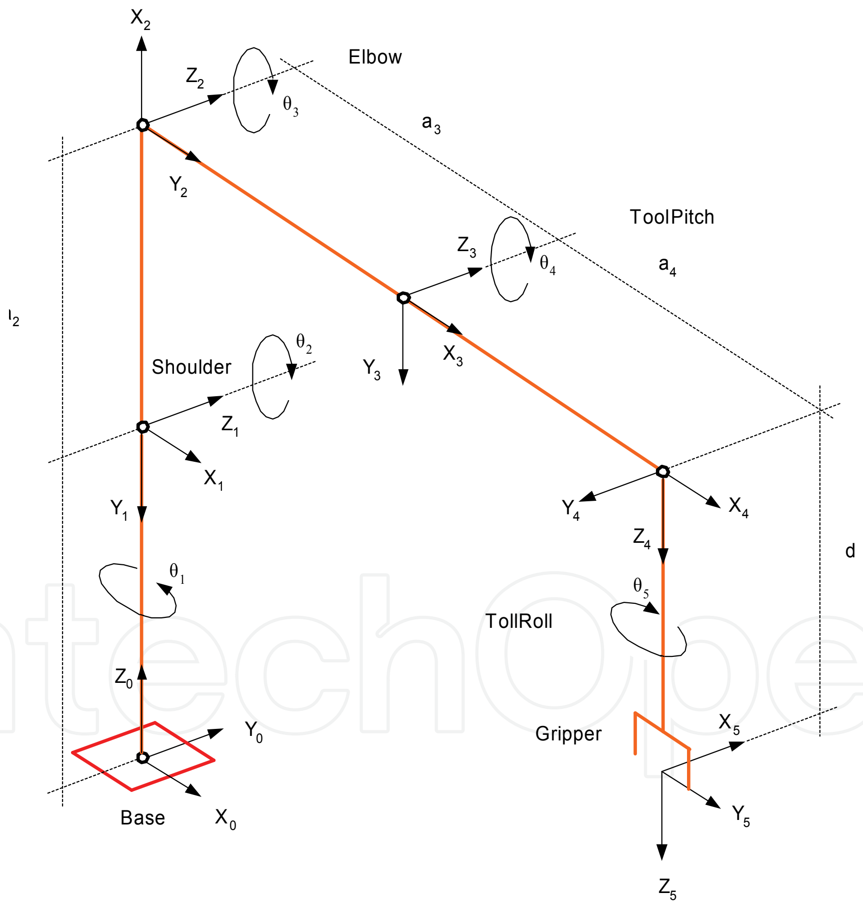

Fig. 8. Kinematical Model for the Rhino XR4 Robot

Only block-type primitives were used to simulate each one of the robot links. For the robot tool three small block primitives were considered to allow simulating the tool opening/closure mechanism. Figure 9 shows the *RobSim* model for the Rhino XR4 at the home position and orientation (initial configuration).

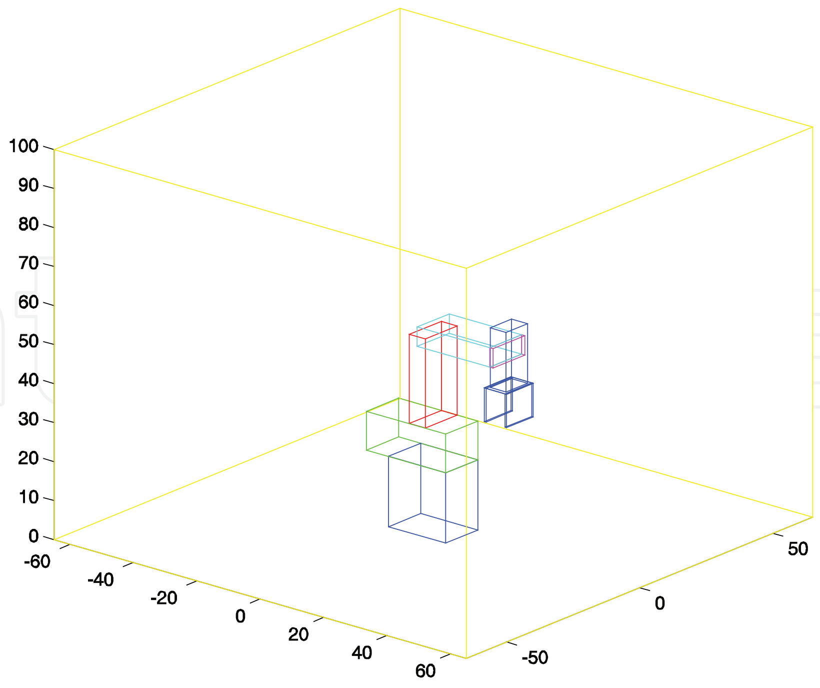

Fig. 9. *RobSim* Model for the Rhino XR4 Robot – Initial Configuration

Figure 10 shows the robot after executing a moving function towards a final configuration.

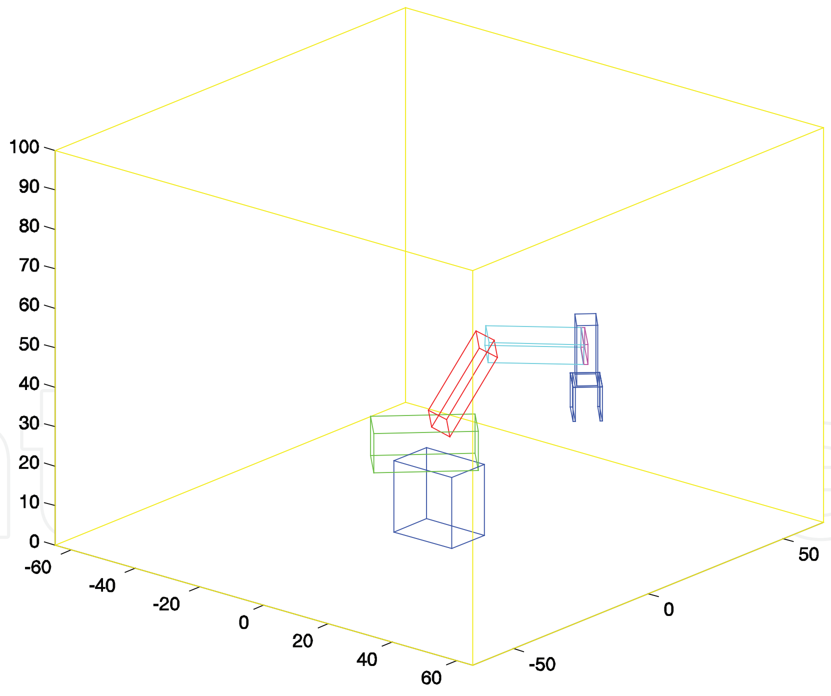

Fig. 10. *RobSim* Model for the Rhino XR4 Robot – Final Configuration

As part of a research project, prototypes of an inspection mobile robot were devised. The *RobSim* platform was particularly suitable to analyze the robots kinematics. The envisaged mobile robot will travel along suspended cables and will execute vision-guided maneuvers in order to overcome obstacles. Figures 11 and 12 show *RobSim* models of two prototypes.

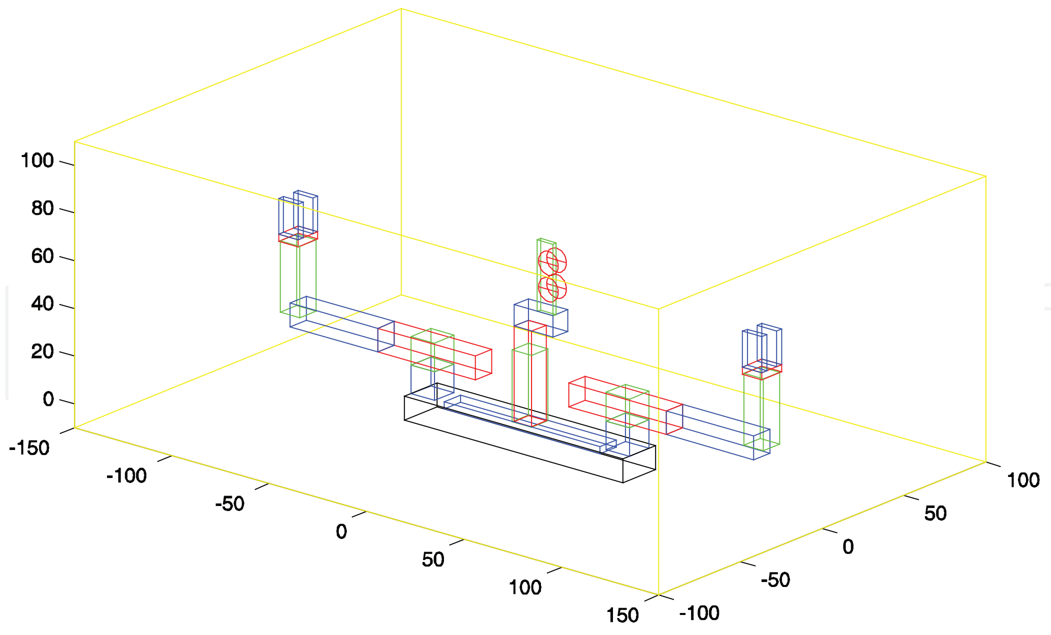

Fig. 11. *RobSim* Model of an inspection mobile robot (Soares & Casanova Alcalde, 2008)

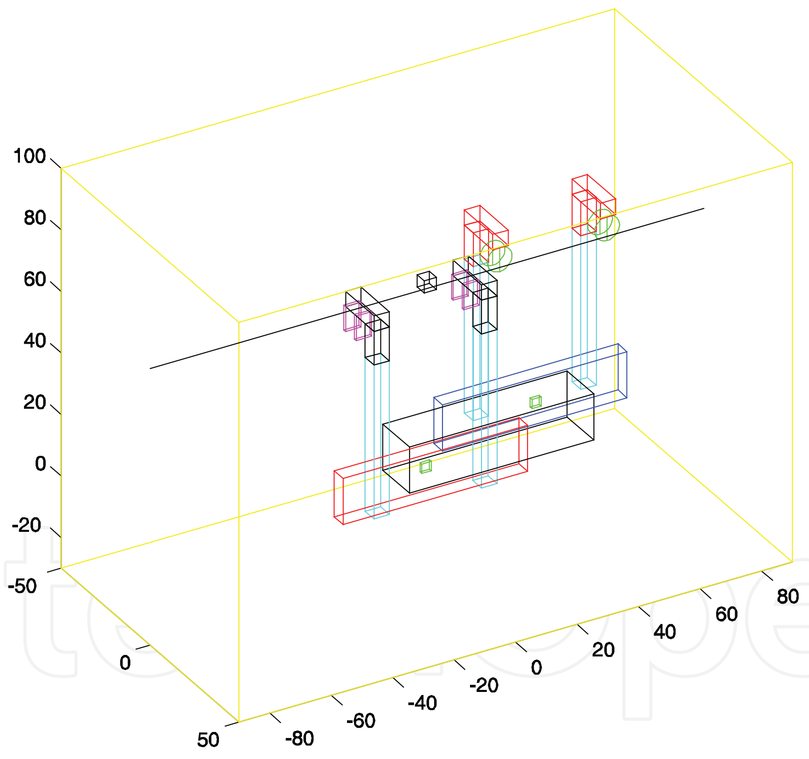

Fig. 12. *RobSim* model of another inspection mobile robot (Soares & Casanova Alcalde, 2008)

5. Visual servo control of robotic systems

Visual servo control of robotic systems uses visual data to implement a feedback control loop to guide the robot in performing a certain task. Therefore the chosen machine vision strategy has to be considered into the robotic system dynamics. The camera for image capture can be mounted on the robot end-effector, or fixed at a certain place to observe the robot workspace. The first approach is called an *eye-in-hand* configuration and the second, an *eye-to-hand* configuration. Other possibilities combining schemes are also possible (Chaumette & Hutchinson, 2007). A variant of the *eye-to-hand* configuration consists on mounting the camera on another robot or on a pan/tilt structure in order to improve the viewing angle. A single camera arrangement for gathering visual data lacks information about depth measurements. Algorithms for position and orientation (pose) estimation could then be introduced or two-cameras can be used to implement a stereo-vision scheme to calculate depth information. This section discusses briefly the main visual-based control schemes. First, a characterization of the control error for a visual servo control strategy is discussed. Then, the position- and the image-based visual servo control schemes are discussed. Some considerations about the system stability are finally pointed out.

5.1 Characterization of the control error for visual servo control schemes

In visual servo control schemes the image coordinates of points of interest are captured. These measurements constitute a set of image measurements represented by $\mathbf{m}(t)$. From these measurements an actual *visual features vector* $\mathbf{s}$ is calculated to represent the actual value of k visual features. It is defined as $\mathbf{s}(\mathbf{m}(t),\mathbf{a})$ (Chaumette & Hutchinson, 2006), where $\mathbf{a}$ is a set of parameters that represent additional knowledge about the system. Vector $\mathbf{a}$ can be an approximation of the camera intrinsic parameters or *3D* models of objects being observed. The desired *visual features vector* is represented by $\mathbf{s}^*$, usually constant, being changes in $\mathbf{s}$ dependent only on camera motion. The objective of the visual servo control is therefore to minimize a *visual features error vector* $\mathbf{e}(t)$ defined by

$$\mathbf{e}(t) = \mathbf{s}(\mathbf{m}(t), \mathbf{a}) - \mathbf{s}^* \tag{5}$$

The visual servo control schemes depend on how the visual features vector $\mathbf{s}$ is determined, as it will be seen in the following subsections. To minimize the visual features error vector $\mathbf{e}(t)$ (Equation 5) a common approach is to implement a velocity controller. Defining the spatial velocity of the camera $\mathbf{V}_c = [\mathbf{v}_c\ \mathbf{\Omega}_c]^T$, being $\mathbf{v}_c$ the instantaneous linear velocity of the origin of the camera frame and $\mathbf{\Omega}_c$ the instantaneous angular velocity of the camera coordinate frame. A relation is then established between the time derivative of $\mathbf{s}$ and $\mathbf{V}_c$

$$\dot{\mathbf{s}} = \mathbf{L}_s.\mathbf{V}_c \tag{6}$$

Where $\mathbf{L}_s$ is a $k\times 6$ matrix related to $\mathbf{s}$ called the image interaction matrix or also a *feature Jacobian*. Assuming a constant $\mathbf{s}^*$ as usual, and using Equations (5) and (6) results in

$$\dot{\mathbf{e}} = \mathbf{L}_s.\mathbf{V}_c \tag{7}$$

A simple strategy could be adopted, for example, an exponential decay of the error ($\dot{\mathbf{e}} = -\lambda.\mathbf{e}$) for a certain $\lambda>0$. Then using Equation (7) and the Moore-Penrose pseudo-inverse matrix $\mathbf{L}_s^+$, $\mathbf{V}_c$ the input of the robot velocity controller will be given by

$$\mathbf{V}_c = -\lambda.\mathbf{L}_s^+.\mathbf{e} \tag{8}$$

For a full rank $\mathbf{L}_s$, the pseudo-inverse will be $\mathbf{L}_s^+ = (\mathbf{L}_s^T.\mathbf{L}_s).\mathbf{L}_s^T$ and $\|\mathbf{V}_c\|$ and $\|\dot{\mathbf{e}} - \lambda.\mathbf{L}_\mathbf{s}.\mathbf{L}_\mathbf{s}^T.\mathbf{e}\|$ will turn to be minimal. For a square matrix $\mathbf{L}_s$, Equation (8) would be $\mathbf{V}_c = -\lambda.\mathbf{L}_s^{-1}.\mathbf{e}$. As in

practice it is impossible to know $\mathbf{L}_s$ and $\mathbf{L}_s^+$, an approximation or estimative, for the pseudo-inverse must be determined, this approximation will be denoted as $\hat{\mathbf{L}}_s^+$.
As mentioned, depending upon the way the visual features vector **s** is established, different visual servoing schemes are possible. Two schemes are considered: a) the image-based visual servo control (IBVS); and b) the position-based visual servo control (PBVS).

5.2 Image-Based Visual Servo control scheme (IBVS)

In this scheme the image features to be determined can be: image-plane coordinates of points of interest, regions of interest of the image, parameters that define straight lines over the image, etc. From these features a visual features vector **s**(**m**(*t*),**a**) is established. Considering the simplest situation, the image measurements vector **m**(*t*) consists of the pixel coordinates of the set of image points of interest. Finally, vector **a** consists of the installed camera intrinsic parameters. In this situation the interaction matrix $\mathbf{L_s}$ can be easily determined. As shown in Figure 4, for a *3D* point ${}^{S_c}\mathbf{P} = [X\ Y\ Z]^T$ referred to S_c, the camera coordinate frame, its projection onto the image plane will be a *2D* point with coordinates ${}^{S_c}\mathbf{p} = [x\ y\ f]^T$, where *f* is the camera focal length. From geometrical relation (Figure 4) *x* and *y* are given by

$$\begin{aligned} x &= \frac{fX}{Z} \\ y &= \frac{fY}{Z} \end{aligned} \tag{9}$$

By using the camera intrinsic parameters (f, p_x, p_y, u_0, v_0, α), *u* and *v*, **p** coordinates referred to the image plane, are given by

$$\begin{aligned} u &= u_0 - \frac{X}{Z}\frac{f}{p_x.\cos\alpha} \\ v &= v_0 + \frac{Y}{Z}\frac{f}{p_y} + \frac{X}{Z}\frac{f.\tan\alpha}{p_y} \end{aligned} \tag{10}$$

From Equation (10), given *X*, *Y* and *Z* it is possible to calculate *u* and *v*. But in the other way round it is not possible to calculate *Z*, the depth of **P** relative to the camera frame.
Time derivatives of *x* and *y* (velocities) in Equation (9) results in

$$\begin{aligned} \dot{x} &= \frac{\dot{X} - x\dot{Z}}{Z} \\ \dot{y} &= \frac{\dot{Y} - y\dot{Z}}{Z} \end{aligned} \tag{11}$$

The *3D* velocity of point **P** (S_c coordinates) is related (Hutchinson et al., 1996) to the camera linear and angular velocities, $\mathbf{V}_c$ and $\mathbf{\Omega}_c$ respectively, as

$$\dot{\mathbf{P}} = -\mathbf{V}_c - \mathbf{\Omega}_c \times \mathbf{P} \tag{12}$$

or

$$\begin{aligned}\dot{X} &= -v_x - \omega_y.Z + \omega_z.Y \\ \dot{Y} &= -v_y - \omega_z.X + \omega_x.Z \\ \dot{Z} &= -v_z - \omega_x.Y + \omega_y.X\end{aligned} \tag{13}$$

Substituting Equation (13) into Equation (11) and with $\mathbf{p} = [x\ y]^T$ results in

$$\dot{\mathbf{p}} = \mathbf{L}_p.\mathbf{V}_c \tag{14}$$

where $\mathbf{L}_p$ is given by

$$\mathbf{L}_p = \begin{bmatrix} -\dfrac{f}{Z} & 0 & \dfrac{x}{Z} & \dfrac{x.y}{f} & -\dfrac{f^2+x^2}{f} & y \\ 0 & -\dfrac{f}{Z} & \dfrac{y}{Z} & \dfrac{f^2+y^2}{f} & -\dfrac{x.y}{f} & -x \end{bmatrix} \tag{15}$$

Matrix $\mathbf{L}_p$ then depends on **P** coordinates, on **p** coordinates and on the camera intrinsic parameters. Any control scheme using this $\mathbf{L}_p$ must estimate *Z*, the depth of **P** relative to the camera frame. Due to $\mathbf{L}_p$ dimension, to control a six axis robot, a minimum of three points will be necessary, so $k \geq 6$. For a visual features vector **s** = ($\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$) three interaction matrixes $\mathbf{L}_{p1}$, $\mathbf{L}_{p2}$ and $\mathbf{L}_{p3}$ must be stacked. To avoid local minimal solutions more than three points are usually considered. For *N* points, $\mathbf{L}_p$ will be a 2*N*×6 matrix.
The main advantage of the IBVS schemes results form the fact that the visual features error is defined only in the image domain, not being necessary any parameter or variables estimation in the *3D* space. A disadvantage is lack of information about the scene depth.

5.3 Position-Based Visual Servo control scheme (PBVS)

In position-based visual servo control schemes the visual features vector **s** is defined using the camera pose (position and orientation) relative to a reference coordinate frame. Determining the camera pose from a set of measurements in one image requires the camera intrinsic parameters and the *3D* model of the object being observed, this is the classical *3D* localization problem. As the PBVS approach needs *3D* reconstruction it is prone to fail due to calibration errors. The general PBVS will not be treated here, only a particular case implemented with a robotic manipulator and a stereo-vision device whose simulation in the *RobSim* platform is reported in Section 6.
From *2D* image data captured by each of a two cameras arrangement (stereo vision) it is possible to reconstruct the *3D* pose of an object in the cartesian manipulator workspace. Once the specification of a desired pose of an object handled by the robot end-effector is given, it is possible to define an error between the actual object pose and the desired one. Since this error is specified in the *3D* workspace and the robot joints are actuated in order to cancel it, this kind of procedure can be considered a position-based control scheme.

5.4 Some considerations about stability

Vision-based control systems have non-linear and highly coupled dynamics. For stability analysis Lyapunov direct method can be applied. A particular Lyapunov function would be

$$V = \frac{1}{2}\left\|e(t)\right\|^2 \tag{16}$$

In case of IBVS, by using Equations (7) and (8) the time derivative of $V(t)$ is

$$\dot{V} = \mathbf{e}^T.\mathbf{e} = -\lambda.\mathbf{e}^T.\mathbf{L}_s.\hat{\mathbf{L}}_s^+.\mathbf{e} \tag{17}$$

A global asymptotic stability is assured if $\dot{V}$ is positive definite or

$$\mathbf{L}_s.\hat{\mathbf{L}}_s^+ > \mathbf{0} \tag{18}$$

If the number of image features k is equal to the camera DOF and a proper control scheme is implemented, then full rank $\mathbf{L}_s$ and $\hat{\mathbf{L}}_s^+$ matrixes will result and the stability condition (Equation 18) will be assured if a well approximated $\hat{\mathbf{L}}_s^+$ is determined (Chaumette & Hutchinson, 2006). But considering a robot with 6 DOF under a IBVS control, where k is usually greater than 6, then the stability condition could never be assured. The resultant $k \times k$ matrix in Equation (18) would have at most a rank of 6, then a nontrivial null space will exist and local minima will result.

6. Visual servo control of a robotic manipulator using *RobSim*

The RobSim platform can help designers to analyze a robotic manipulator under a control scheme. To illustrate this approach a visual servo control scheme is applied to a robotic workstation consisting of the Rhino XR4 robot and a computer vision device. Visual servo control uses visual information to control the pose (position and orientation) of the robot end-effector in order to perform a specified task.

6.1 An image-based visual servoing scheme within *RobSim*

For camera simulation within the RobSim platform it is necessary to set up the camera primitive (Section 3), i.e. introduce the camera intrinsic and extrinsic parameters into its initialization, moving and displaying functions. Using the perspective projection model (Hutchinson et al., 1996) two reference frames are of concern: the camera reference frame, Sc, and the sensor reference frame, Ss. The camera reference frame is the one attached to the primitive camera as shown in Figure 3. Given a point P, represented in the Sc frame as ${}^{S_c}\mathbf{P} = [X\ Y\ Z]^T$, its *2D* projection point $\mathbf{p}$ onto the image sensor plane referred to the S_s frame will be, in homogeneous coordinates, ${}^{S_c}\mathbf{p}_h = [u\ v\ 1]^T$, being its pixel coordinates calculated from Figure 4. Executing the *RobSim* image acquisition function $\mathbf{p}_{imag}$=*point_view*($\mathbf{p}_{3D}$,$\mathbf{K}_i$,${}^0\mathbf{T}_c$) (Subsection 3.4) is possible to simulate a (Chaumette & Hutchinson, 2006)point capture as the camera moves. The $\mathbf{p}_{3D}$ vector, a workspace point relative to the base coordinates, is measured in centimeters. The $\mathbf{p}_{imag}$ vector, the *2D* corresponding point onto the image plane, is measured in pixels.

The *RobSim* features for visual servo control will be shown in a vision-guided operation with the Rhino XR4 robot. Figure 13 shows the robot *RobSim* model at its home pose (initial configuration) with a camera attached to its end-effector (gripper), so with the 5 DOF motion capability the robot allows. Resting over the base plane there is a cube (a block primitive) with color marks (asterisks) at its four top vertexes. Figure 13 also shows a window displaying the cube image as captured by the camera, in which the cube is represented by the four top color marks. An additional mark represents the image plane center.

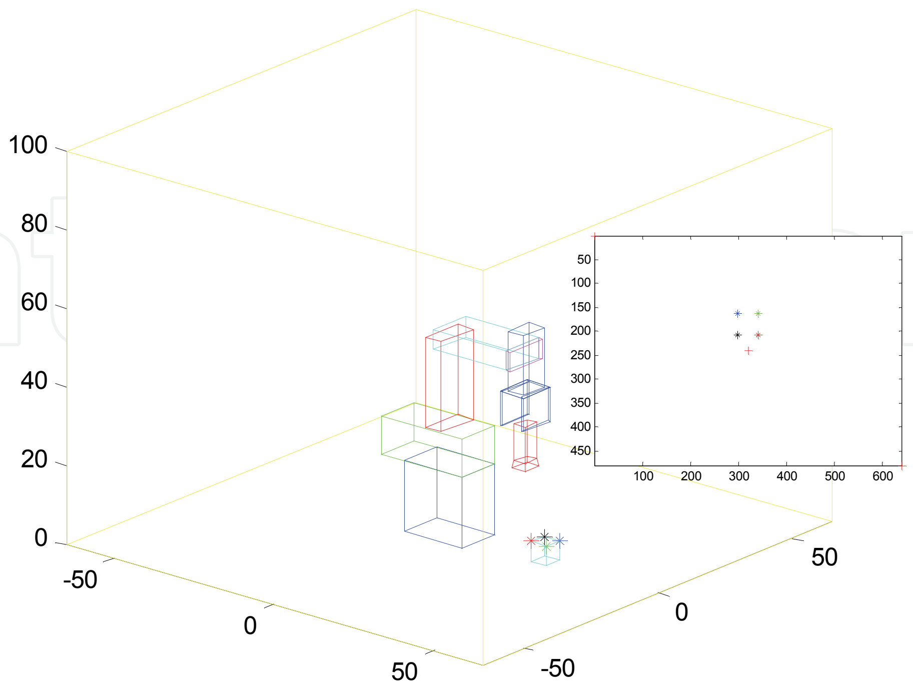

Fig. 13. *RobSim* vision-guided operation – initial configuration

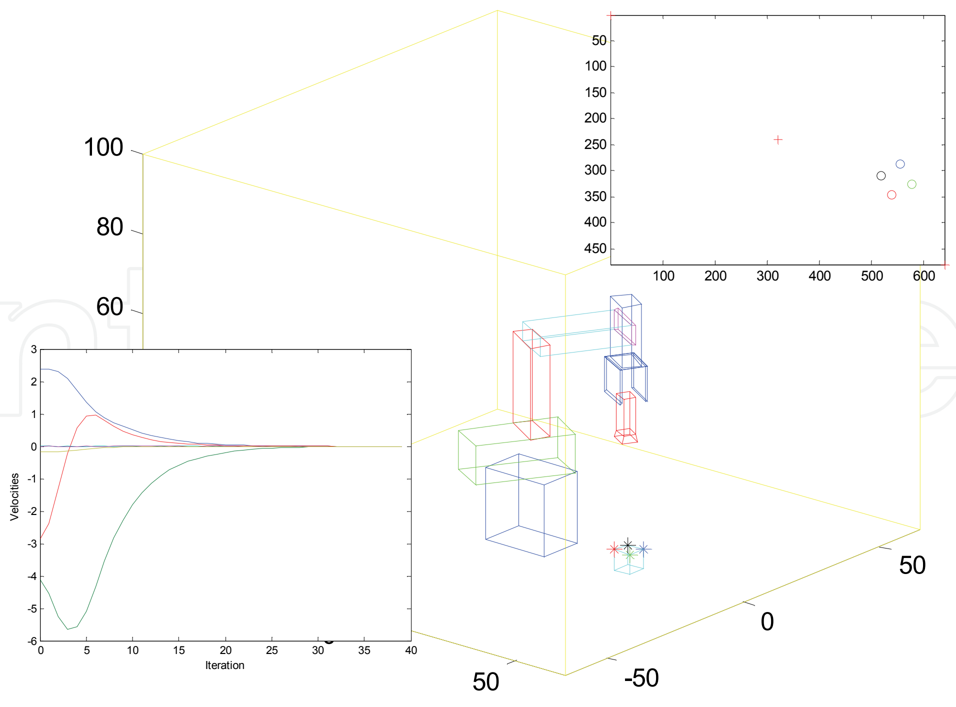

Fig. 14. *RobSim* vision-guided operation – new configuration

In this image-based servoing scheme the visual features error vector is defined as the difference between current and desired cube vertex positions. An exponential decoupled decay for this error was imposed by a velocity control policy. Camera reference velocities were then obtained using the image interaction matrix. In turn, the joint reference velocities for the robot joints controllers were obtained from the robot Jacobian.
Figure 14 shows the robot after executing a moving command towards a new configuration while the cube remains fixed. The window image shows the cube image, represented by the correspondent color marks (now circle marks). Another window shows the time variations of the camera velocity components. Visual information can be then used to guide the robot to describe a trajectory from an initial configuration to a new configuration through individual joint control.

6.2 A position-based visual servoing scheme within *RobSim*

Here, the PBVS architecture was implemented to simulate a vision-guided placing operation with the Rhino XR4 robot and a stereo-vision system with two cameras in the robot workspace. The object to be handled is a cube represented by a block-type primitive. Three marking points are located at three vertexes of the cube in order to visually represent the cube for translation and rotation displacements. Figure 16 shows the initial configuration of the robotic manipulator with the cube being grasped by the end effector, the cube initial pose (green) and the cube final pose (cyan).

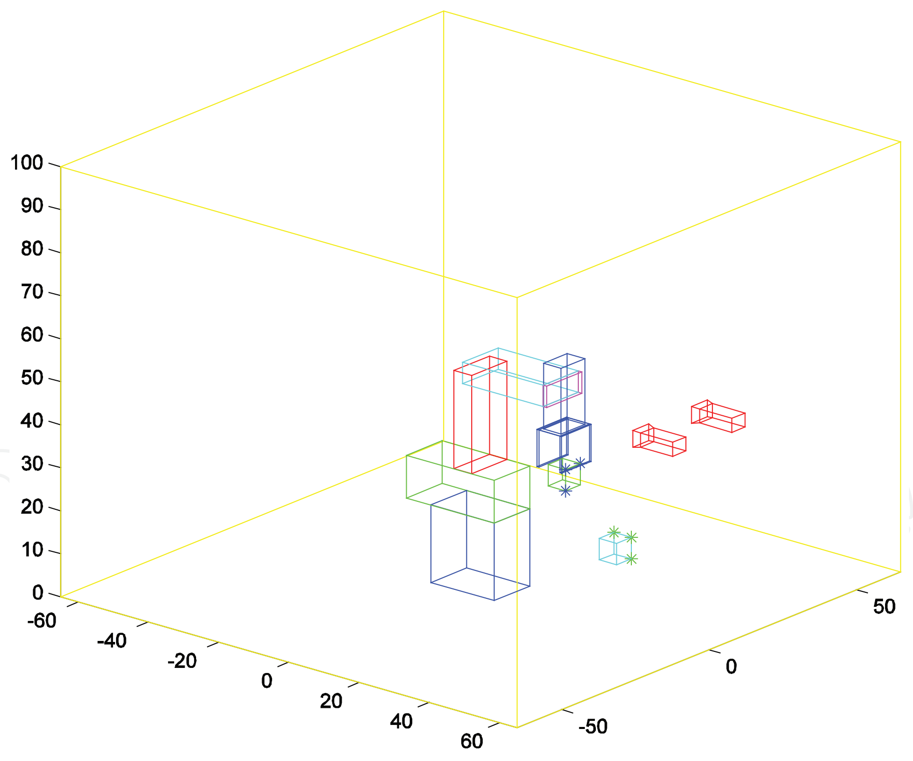

Fig. 15. Vision-guided placing operation – initial configuration

A computer vision algorithm is not required in this case because the object is synthetic and a simple one. Determination of the coordinates of the three vertexes that identifies the cube is performed by the stereo-vision system (Hutchinson, 1996). The coordinates of the three identifying vertexes representing the cube at its initial pose are, $\mathbf{p}_{a1}$ (middle vertex), $\mathbf{p}_{b1}$ and

$\mathbf{p}_{c1}$. The corresponding three coordinates at the final pose are $\mathbf{p}_{a2}$, $\mathbf{p}_{b2}$ and $\mathbf{p}_{c2}$. From these points four *3D* vectors are generated: $\mathbf{P}_{ab1}$ pointing from $\mathbf{p}_{a1}$ to $\mathbf{p}_{b1}$; $\mathbf{P}_{ac1}$ pointing from $\mathbf{p}_{a1}$ to $\mathbf{p}_{c1}$; $\mathbf{P}_{ab2}$ from $\mathbf{p}_{a2}$ to $\mathbf{p}_{b2}$; and finally $\mathbf{P}_{ac2}$ from $\mathbf{p}_{a2}$ to $\mathbf{p}_{c2}$. All these vectors are normalized before use.

To describe the robot joint dynamics a first-order model without dissipation is considered. Once the end-effector velocity vector $\dot{\mathbf{r}}(t)$ (translational and rotational motion) referred to the base frame coordinates is known, the robot inverse kinematics model can be used to determine the joint velocities vector $\dot{\mathbf{q}}(t)$ (Schilling, 1990). These velocities vectors are related by the pseudo-inverse of the robot Jacobian matrix, $\mathbf{J}(\mathbf{q})$ as

$$\dot{\mathbf{q}}(t) = \mathbf{J}^{+}(\mathbf{q}).\dot{\mathbf{r}}(t) \tag{19}$$

The end effector velocity $\dot{\mathbf{r}}(t)$ is known as the *screw* velocity, consisting of a linear velocity along a line and an angular velocity around that line. Its first three elements are the linear velocities $\mathbf{T}_r = [v_x\ v_y\ v_z]^T$ and its last three elements $\mathbf{\Omega}_r = [\omega_x\ \omega_y\ \omega_z]^T$ the angular velocities, being all components referred to the base coordinate frame. Thus, the end effector velocity is

$$\dot{\mathbf{r}}(t) = [\mathbf{T}_r\ \mathbf{\Omega}_r]^T \tag{20}$$

A task function characterizing position and orientation errors of the cube handling task was implemented. By vector analysis, it can be shown that if $\mathbf{P}_r = (\mathbf{P}_{ab1}\times\mathbf{P}_{ac1})\times(\mathbf{P}_{ab2}\times\mathbf{P}_{ac2}) = \mathbf{0}$ (where × denotes vector cross product), the handled cube attains the reference or desired orientation, in the particular cases where $\mathbf{P}_{ab1}$ and $\mathbf{P}_{ab2}$ or $\mathbf{P}_{ac1}$ and $\mathbf{P}_{ac2}$ have the same direction. The angular control velocity is adjusted as $\mathbf{\Omega} = \mathrm{k}_1\mathbf{P}_r$, where k_1 is a positive proportional gain.

It is also verified that, being $\mathbf{t}_a$ a vector from point $\mathbf{p}_{a1}$ to point $\mathbf{p}_{a2}$ and $\mathbf{p}_{a1v}$, a vector from the frame origin to point $\mathbf{p}_{a1}$, the vector $\mathbf{P}_t = \mathrm{k}_2\mathbf{t}_a + \mathbf{\Omega}\times\mathbf{p}_{a1v}$, with k_2 a positive proportional gain, is equal to the null vector when the handled cube assumes the reference pose. In this case the translation control velocity is given by $\mathbf{T}_r = \mathbf{P}_t$. By adequately adjusting k_1 and k_2 it is possible to improve the regulation velocity of position and orientation errors.

Figure 16 shows the final configuration of the vision-guided placing operation, a window shows the initial image as seen by the left camera. Another window shows the time evolution of the end-effector velocity components (Equation 20), in which case, due to the initial and desired cube pose, the angular components ω_x and ω_y are zero.

7. Conclusion

A software platform *RobSim* for analysis and design of robotic systems that includes image capturing devices was presented. It was developed within the Matlab environment to simulate kinematics of robotic structures and it allows implementing control strategies in order to follow trajectories, perform tasks, etc. Thus it is very suitable to implement robotic experiments before dealing with the real system. The platform is based on basic units called *primitives* that assembled together can simulate any robotic structure. Being modular it is expandable, another advantage is the inclusion of a video capturing device that allows implementing vision-guided robotic experiments. The platform was used here to model and simulate fixed and mobile robots. Image- and position-based servoing schemes were implemented for a robotic manipulator with a single and a two-camera arrangement and

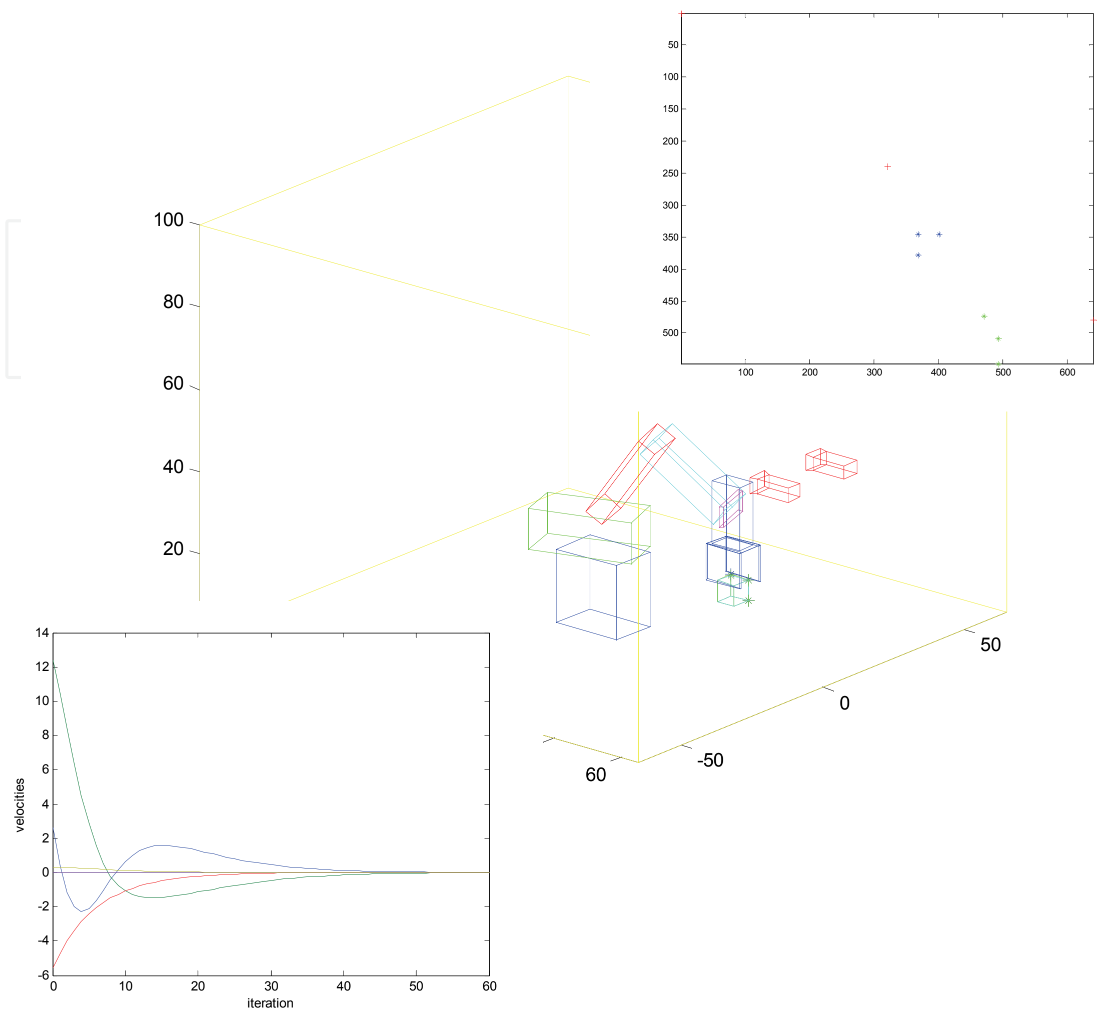

Fig. 16. Vision-guided placing operation – final configuration

simulations carried out within the *RobSim* platform. Further work is being addressed to introduce dynamical parameters into the primitives and simulation of more complex image features acquisition rather than image points.

8. References

Cervera, E. (2003) Visual Servoing Toolbox for Matlab/Simulink, http://vstoolbox.sourceforge.net/

Corke, P. I. (1996), A Robotics Toolbox for Matlab, *IEEE Robotics and Automation Magazine*, Vol. 3, No. 1, pp. 24-32.

Chaumette, F. and Hutchinson, S. (2006), Visual Servo Control – Part I: Basic Approach, *IEEE Robotics and Automation Magazine*, Vol. 13, No. 4, December 2006, pp. 82-90.

Chaumette, F. and Hutchinson, S. (2007), Visual Servo Control – Part II: Advanced Approaches, *IEEE Robotics and Automation Magazine*, Vol. 14, No. 1, March 2007, pp. 109-118.

Hutchinson, S.; Hager, D. & Corke, P. I. (1996), A Tutorial on Visual Servo Control, *IEEE Transactions on Robotics and Automation*, Vol. 12, No. 5, October 1996, pp. 651-670.

Legnani, G. (2005) Spacelib – Package for Matlab – User's Manual: http://www.bsing.ingunibst.it/glegnani.

Schilling, R. (1990), *Fundamentals of Robotics – Analysis and Control*, Prentice-Hall, ISBN-10: 0-1334-4433-9, NJ, USA.

Soares Jr., L. R. & Casanova Alcalde, V. H. (2006), Building Blocks for Simulation of Robotic Manipulators, *Proceedings of the 13th IASTED International Conference on Robotics and Applications*, pp. 408-413, ISBN 978-0-88986-685-0 , Würzburg, Germany, September, 2006.

Soares Jr., L. R. & Casanova Alcalde, V. H. (2006), An Educational Robotic Workstation based on the Rhino XR4 Robot, *Proceedings of the 36th ASEE/IEEE Frontiers in Education Conference*, pp. 7-12, ISBN 1-4244-0257-3 , San Diego, CA, USA, October 28-31, 2006.

Soares Jr., L. R. & Casanova Alcalde, V. H. (2008), *A Robotic Vehicle for Inspection and Maintenance Tasks over Transmission Lines*, University of Brasilia Technical Report.