Real-Time Robotic Visual Servoing using Reinforcement Learning Agents

Marco A. Pérez-Cisneros, Alberto De-la-Mora-Galvez
Centro Universitario de Ciencias Exáctas e Ingenierías, CUCEI
Universidad de Guadalajara
Blvd. Marcelino García Barragan, Guadalajara Jalisco MEXICO 44860
Email: marcopc@cucei.udg.mx demora@redudg.udg.mx

Abstract—This paper presents an open platform to design, simulate and test robotic image-based visual servoing systems using off-the-shelf commercial cameras. The objective is to investigate the use of reinforcement learning (RL) algorithms such as an adaptive critic to increase the overall performance. First a set of simulation files for Matlab is introduced and then a real-time toolkit for visual servoing is presented. The classical RL actor-critic architecture performs on-line adjustment of the linear trajectory regulator in order to achieve real-time improvement and adaptation without losing an acceptable regulation. The RL system learns directly from data in the image space and the current state of the robot. Two feedforward neural networks are used, with the actor network performing the adjustment whereas the adaptive critic is storing and assigning action values. Both toolkits are used to design an eye-in-hand image-based visual servoing system for real-time tracking in the TQ MA2000 robot which exhibits high backlash and low resolution. Simulation and real-time results are documented in the paper.

I. INTRODUCTION

Different implementations of Visual Servoing (VS) have been traditionally identified within two main classifications: position-based and image-based visual servoing. An introductory overview of both classes is presented in the now classic VS tutorial in [1]. Basically, in the image-based visual servoing (IBVS) the error signal is computed in the image plane and the regulation commands are generated with respect to such error by means of a visual Jacobian. On the other hand, in the position-based schemes, the image features are used to estimate an object-workspace characterization in such a way that the error can be computed in the Cartesian space and used in the control loop.

Traditionally image-based systems have been regarded to possess a good robustness to calibration errors [2], [3], even in the absence of the object and workspace model. However image-based VS schemes also exhibit some weaknesses such as singularities in the visual Jacobian which may lead to conflicts in the control loop. Other major drawback resides in the fact that an image-based system does not control the robot’s end-effector in the Cartesian space, sometimes resulting in complicated or even unrealistic joint configurations being demanded to the robot. New IBVS schemes have been proposed to avoid these problems. Some of the new schemes combine 2-D and 3-D information and pose estimation to create a more complete visual servoing algorithm, for instance the 2-1/2 Visual servoing [4]. However these servoing schemes often require the description of object or its visual features. Usually 2-D visual servoing schemes can not cope with objects which are not acceptably described which limits the applicability of servoing systems to those cases where the object of interest either is difficult to characterize or an exact description is not available due to alterations of the original model as a result of damages or accidents.

Notice as well that many VS schemes assume no significant mechanical problems, low latency and high resolution in the driven robot, which is not always the case. This occurs specially in old, low-cost 6-DOF manipulators which normally exhibit limited repeatability, low accuracy and sometimes high mechanical backlash. In this paper, a low-profile anthropomorphic planar manipulator is equipped with a commercial CCD camera attached to its end-effector. An IBVS algorithm is used to achieve low-speed tracking of a moving object, which is model-free in the sense that no object model is provided [5]. Like other tracking problems, the aim is to keep the camera in a plane parallel to the tracked object. The objective is to investigate the use of RL algorithms to increase the performance of the visual tracking task. The heuristics is incorporated into the system by designing an actor-critic system [6] which evaluates the performance in the image space, i.e. directly considering the location of the visual features in the image with respect to their target locations.

The classical RL actor-critic scheme performs on-line adjustment of the parameters driving the linear trajectory regulator. By using the adaptive heuristic critic (AHC) approach [7], the training aims to achieve real-time improvement and adaptation without losing an acceptable regulation of the visual servoing task. The RL system learns directly from data in the image space and the robot current state. Two feedforward networks are used, the actor directly modifies the regulator gains and the adaptive critic stores and assigns action values.

The following sections contains a description of the IBVS tracking system. Special emphasis is given to the simulation toolkit for Matlab and the real-time toolkit. The computer vision part of the real-time toolkit deserves special attention because it also interfaces to the Intel’s Open Computer Vision library which results in a standard and portable user front-end. Also the paper presents a brief overview of the RL schemes to introduce a detailed description of the learning algorithm used in the experiments. As pointed out before, the whole library
has been successfully applied to design and test a real-time visual tracking task with a 6-DOF robot which is equipped with a commercial off-the-shelf low-profile CCD camera. The results of the simulation are compared to performance graphs of the system working with no heuristic adjustment. The exposition finalizes by discussing some guidelines for future development.

II. THE SIMULATION TOOLKIT

The simulation toolkit for Matlab/Simulink used in the experiments keeps full compatibility with the Robotic Toolbox [8], the Camera Calibration toolbox, and several object and camera classes taken from [9]. The toolkit actually provides a more proper framework to integrate all these well known packages offering practical solutions to common visual servoing issues. Relevant Matlab commands are generally also available as Simulink blocks. Each task is grouped together with other related functions which facilitates the creation of operational blocks. The set of visual servoing simulation includes object models which can be easily incorporated into trajectory generators to simulate moving bodies. Also any camera of known geometry can be simulated using the imaging model block. An S-file provides the visualization of the camera screen and includes the recording of visual features. The visual Jacobian and its inverse are also included because they belong to the core of any IBVS. Although a PI regulator is employed in this paper, several other visual regulators can easily be built.

III. THE REAL-TIME TOOLKIT

The real-time set of commands is compacted into one exportation library. One of its major features is a number of special purpose data structures which are used by convention in the library. They are the foundations of the modular design philosophy behind the toolkit. They are organized by layers covering from elemental vector representation to dynamically computed visual Jacobian.

A selection of the most significative functions in the real-time toolkit is presented in Table I together with a simple description. Notice those functions for timing a real system and the functions for matrix algebra. It is easy to identify a group of functions devoted to robot Kinematics and the group for visual computation. Some of the commands to implement network services and digital filters are also shown. In the next section, both toolkits are used to design, simulate, build and test the tracking task example. The system’s step response is used to analyze the performance.

IV. MODELLING THE IBVS EXPERIMENT

One of the objectives is to use a commercial off-the-shelf low-profile CCD camera as visual sensor. In our setup, one Logitech Quick Cam Pro 4000 CCD camera is attached to the last actuator of the TQ MA2000 robot. This commercial camera can provide high-quality and low-noise images at low range of 30fps. The intrinsic and extrinsic camera parameters were experimentally determined by using the camera model and calibration method created by Heikkilä and Silven in [10]. The camera model is compacted into the $C$ matrix as follows

$$C = \begin{bmatrix} f_u & \alpha_c f_u & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 408.99 & 0 & 187.86 \\ 0 & 409.68 & 129.17 \\ 0 & 0 & 0 \end{bmatrix}$$

with $f_u$ and $f_v$ being the focal distance in pixel units whereas $(\alpha_c \cdot f_u)$, $u_0$ and $v_0$ are the camera intrinsic parameters. The matrix $C$ is used to calculate the location of an image point $(u_i, v_i)$ following the so-called camera pin-hole model, i.e. defining its metric value $(x_i, y_i)$ in the camera coordinate frame. A smaller matrix $A$ is a sub-matrix of $C$ with the image centers $(u_0, v_0)$ directly discounted from the pixel values as follows

$$J_v(s_t, Z_t, A) = \begin{bmatrix} \alpha_u \\ 0 \end{bmatrix} \begin{bmatrix} \alpha_w & 0 \\ 0 & \alpha_v \end{bmatrix} \cdot \begin{bmatrix} -\frac{1}{Z_i} \\ 0 \end{bmatrix} = \begin{bmatrix} -\frac{1}{Z_i} \alpha_u \alpha_w f_u \\ -\frac{1}{Z_i} \alpha_v \alpha_w f_v \end{bmatrix} \begin{bmatrix} x_i y_i \\ -x_i y_i \end{bmatrix} = \begin{bmatrix} -\frac{1}{Z_i} \alpha_w f_u \alpha_v y_i \\ -1 + x_i^2 \end{bmatrix} \begin{bmatrix} x_i y_i \\ -x_i y_i \end{bmatrix}$$

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>basetime</td>
<td>Time-base computation.</td>
</tr>
<tr>
<td>buildExch</td>
<td>Builds the (6x6) exchange matrix</td>
</tr>
<tr>
<td>builtT</td>
<td>Builds the transform matrix with kinematic parameters.</td>
</tr>
<tr>
<td>clearJ</td>
<td>Clear memory to hold (6x6) JACOBIAN matrix</td>
</tr>
<tr>
<td>compO</td>
<td>Computes the “O” column using columns “N” and “A”.</td>
</tr>
<tr>
<td>computeJ</td>
<td>Robot Jacobian</td>
</tr>
<tr>
<td>computeJ0</td>
<td>robot Jacobian wrt base frame</td>
</tr>
<tr>
<td>controller</td>
<td>Trajectory generator regulator</td>
</tr>
<tr>
<td>crossp</td>
<td>Cross product two vectors.</td>
</tr>
<tr>
<td>cutting</td>
<td>Filters out numerical noise</td>
</tr>
<tr>
<td>desiredT</td>
<td>Builds target HT matrix.</td>
</tr>
<tr>
<td>extractInt</td>
<td>Extracts features from the network receptor.</td>
</tr>
<tr>
<td>fkinematic</td>
<td>Solves robot kinematics</td>
</tr>
<tr>
<td>JacVect</td>
<td>Prepares screw vector with the gain and integrates it.</td>
</tr>
<tr>
<td>mMatrix</td>
<td>Square matrix product</td>
</tr>
<tr>
<td>multJ</td>
<td>Two Jacobians product</td>
</tr>
<tr>
<td>prodJ</td>
<td>HT matrix product.</td>
</tr>
<tr>
<td>screwTMat</td>
<td>Transforms Jacobian wrt camera frame</td>
</tr>
<tr>
<td>Tdiff</td>
<td>Computes the differential vector</td>
</tr>
<tr>
<td>vErrorSub</td>
<td>Error feature vector</td>
</tr>
<tr>
<td>visionCx</td>
<td>PI visual controller</td>
</tr>
<tr>
<td>visionJac</td>
<td>Computes visual Jacobian</td>
</tr>
<tr>
<td>visionPlInv</td>
<td>Inverse visual Jacobian</td>
</tr>
<tr>
<td>visionPrep</td>
<td>Calculate metric coordinates</td>
</tr>
</tbody>
</table>

TABLE I

SELECTED FUNCTIONS FROM THE REAL-TIME VS TOOLKIT.
As shown in [11], the visual Jacobian is now a function of the so-called metric pixel coordinates of each feature $s_i = [x_i, y_i]^T$, the depth estimation $Z_i$ and the camera matrix $A$, taking the form of expression 1. It allows an improved image-based visual servoing method as discussed in [12]. Notice that two rows like those in Equation 1 are defined for each feature in the image. The final visual Jacobian is thus conformed by stacking these rows. In a typical IBVS implementation with four tracked features, the visual Jacobian dimension is $(8 \times 6)$ with six corresponding to the number of the robot’s DOF.

The computation of the inverse visual Jacobian is required to calculate the manipulator motion in response to changes in the image features. The velocity screw is calculated under the task function approach [13] as $v = -\alpha \cdot J^*_x \cdot e$ with $e$ being the error vector between the current feature vector $f$ and the target $f^*$. The inverse Jacobian matrix $J^*_x$ is calculated by the pseudo-inverse matrix. In order to facilitate the design and the integration of components of different nature in a common project, the well-known Robotic toolbox for Matlab [8] is used together with the Visual Servoing toolkit [14] to create a detailed model of the visual tracking task which includes an object in motion over a circular trajectory, in front of the robot’s base.

V. ACTOR-CRITIC LEARNING ARCHITECTURE

The main inconvenience of using Reinforcement Learning (RL) on real-time control schemes is the necessity of giving time to the learner as to receive feedback from the environment and as to learn. This does not always is compatible to the time constraints imposed by a real-time system. RL resides in the middle of supervised and unsupervised algorithms. After receiving the system state, the learner receives a reinforcement from the the environment notifying about the usefulness of its output. The main objective is therefore to maximize this reward signal over the time. This can be achieved by trial and error training until the learner is able to discover those outputs with a maximum reward. RL matches well with a control environment [15], because the learner is able to optimize over the time and hence to minimize a given error tracking policy, for instance the mean of the sum of square errors (MSSE) in the tracking. Remarkably a RL scheme is naturally capable of dealing with system latency or delays and time constraints. Moreover a RL algorithm posses the ability to explore new states and to exploit past knowledge. So the learning algorithm should also be in charge of guiding the learning to equilibrate the operational modes of exploration and exploitation.

The solution proposed in this paper, goes back to early efforts in RL architectures to use an actor-critic design. Basically, this scheme has two networks, one to perform the parameter adjustment, named the Actor and one to learn from rewards, known as the critic. This classic architecture, widely discussed in [6], is able to meet real-time demands because both the learner and the controller are implemented in one entity. However its main drawback is the complexity of the training algorithm which is alleviated in this paper by using a backpropagation-like procedure. The learning in the evaluation network is based on temporal differences [6], specifically in the adaptive heuristic critic (AHC) approach. The original algorithm explained in [7] had to be modify before its inclusion in the IBVS control loop.

A. Description of the learning system

Figure 1 shows a block representation of the overall system. The experimental plant is represented by the MA2000 Robot model and two regulators, the trajectory generator and the robot’s position controller. The internal feedback of the state is used by the position controller. The arrow crossing the trajectory generator block represents the adjustment process performed by the action neural network. The model of the camera attached to the robot and the visual processing are considered in the two blocks above the robot’s block. Also the IBVS components are represented in two blocks, the visual Jacobian and the visual linear regulator which uses a classical

$$
A = \begin{bmatrix}
  f_u & (\alpha_c \cdot f_u) \\
  0 & f_v
\end{bmatrix} = \begin{bmatrix}
  \alpha_u & \alpha_{uv} \\
  0 & \alpha_v
\end{bmatrix} \cdot \begin{bmatrix}
  u_i - u_0 \\
  v_i - v_0
\end{bmatrix}
$$

$$
x_i = \begin{bmatrix}
  \frac{1}{\alpha_u} - \frac{\alpha_{uv}}{\alpha_v} \\
  0
\end{bmatrix} \cdot \begin{bmatrix}
  u_i - u_0 \\
  v_i - v_0
\end{bmatrix}
$$

![Fig. 1. Block diagram of the IBVS for tracking with RL actor-critic support.](image-url)
PI algorithm. The blocks in the upper left corner represent the actor-critic agents. The critic is influenced by a composite reinforcement signal formed by several operations as explained in [16]. The inputs to both neural networks include two normalized vectors: the robot’s state vector and the image performance vector. The normalization suppresses the scaling problem derived from the differences in magnitude of the input signals. The training of the actor network is governed by the internal evaluation (v) provided by the critic.

The pixel error vector is the average of the difference of each feature with respect to its correspondent desired location in the image. Given that the camera is to move in a plane parallel to the object, and that the features ideally do not move in the object, it can be assumed that this error as constant for all features.

### B. The actor and critic agents

Both networks, the Evaluation and the Critic network, are based on a three layer feedforward distribution which includes a direct bypass of the input vector in order to augment the mapping ability of the network (see Figure 2). This architecture, first used in experiments aiming to find optimal strategies for problem solving [7], uses the same number of hidden and input neurons. The main feature in the nets in Figure 1 is that the critic network’s output is only a weighted average whereas the actor network includes a logarithmic sigmoid transfer function at the output given the probabilistic search performed by the agent [16].

The output of the evaluation network -also named the critic, is the estimated action value assigned to that state. Recall again that an action-value in RL is the probability of maximizing the total reward starting from that state. In the case of the actor agent, the network’s output is put through a logarithmic sigmoidal function which results in a probability function whose values fall between 0 and 1. This value represents the stochastic estimation of the required adjustment value. If the actor-critic scheme in this paper was to be used in solving deterministic systems with multiple actions at a time, then a simple competition between the probabilities for different actions will point out the best to follow [7]. This is not the case for the IBVS, because only one output is required, i.e. an increase or decrease in the gains of the IBVS trajectory regulator. A method to balance this dilemma is to apply a fair competition between different actions corresponding to increase or decrease the gains, winning the option with the higher probability of success. Any selection thus affects the gain vector in the IBVS and is sequentially evaluated by the critic which determines proper adjustments in case of undesired behavior or a positive “boost” for satisfactory responses. So the only practical information about the performance of the learning system is the reinforcement signal. A complete review on the learning algorithm can be found in our previous work in [16].

### VI. Simulated and Real-time Experiment

In the simulation the object is moving on a circular trajectory with \( r = 0.44 \) m and a constant angular velocity of \( \omega = 0.112 \). So the target, a small replica train, is able to ride the arc of 45 degrees in approximately 13.99 seconds. Although different sample times are involved in a IBVS scheme [2], the simulation employs the image processing sample time of 0.033 ms which practically limits the overall real-time bandwidth to fall between 1.5 and 7.5 Hz.

In the simulated experiment, one epoch lasts until a failure signal appears or the train has reached the end of the 45° arc. Given the exploratory behavior of the RL scheme, failure signals are normally expected at early training stages, forcing the system to abort the run and to end the epoch. In that case, the state variables of the robot and target object must be reset and the simulation re-starts with the robot manipulator at the initial position and the object within the camera’s field of view. Notice however that the network weights are updated and stored according to the last evaluation signal which includes the failure state. The training is conducted for 10 epochs at most. If the system is able to complete 3 epochs with no failure states, the learning phase is considered complete and the network weights are saved to be applied to the real system during the operational phase.

The parameter values for the learning algorithm used in the simulation are \( \alpha_h = 0.2 \) and \( \alpha = 1.0 \) for the action network and \( \beta_h = 0.05, \beta = 0.2 \) for the critic net. By using these values, successful learning was accomplished in most cases. On the other hand, the temporal difference gain in the composite reinforcement constructor is \( \varphi = 0.9 \). This greedy value speeds up the convergence process and matches with other RL experiments reported in the literature. The initial gain vector for the trajectory generator has -1.75 for each of its six components, which is a low value with respect to the range of valid gains, experimentally determined to fall between -1.0 and -5.0.
A. Real-time Experimental Setup

After the initial design and training stage, the IBVS system with the support from the RL neurocontroller was tested on the MA2000 robotic experimental setup presented in [14]. The performance is analyzed through the step-response of the VS tracking system. Initially the robot and the camera are located over the moving object following the teaching-by-showing technique. The object is a small replica train. The step response experiment begins with the train moving from its stationary position at a constant velocity of 3.65 cm/sec, following a straight trajectory. As the robot starts tracking, the object stops after 2 seconds allowing the robot to completely catch the object as its features return to the target location in the image space.

The trajectory regulator was initialized with $K = -2.0$, instead of the $K = -1.75$ used in the simulation, only to allow a better comparison with previous performance results from the nominal system. The gains in the PI visual regulator are $K_p = 2.0$ and $K_i = 0.1$, solving the joint space by means of a rate-controlled integrator.

VII. RESULTS AND DISCUSSION

The complete system was simulated at least ten times with random initial values in the weights of both networks. The simulation results in this paper correspond to the third run of the system. After four epochs, the neuro-controller (the actor feedforward network) is able to increase the quality of the tracking task. Figure 3(a) shows the trajectory of the features in the image plane for the nominal linear system represented by the asterisks and for the RL-supported system represented with smaller dots. It is evident that under the neural influence, the error between the target position and the visual features has been reduced as the object moves. Moreover the usual circle-like trajectory of the features in the image [12] is changed as a result of the heuristic contribution of the RL agent. Figure 3(b) shows the evolution of the trajectory generator gain which is directly modified by the action network.

In practical experiments, the overall performance can be simply evaluated by comparing the trajectory of the features in the image space using the nominal controller and the RL-supported control structure. The response from the nominal controller is shown in Figure 4(a) whereas Figure 4(b) shows the response of the RL-supported control structure.

Although both controllers are able to perform the task, the overall error in the image space using the linear controller is higher. The improvement price however is that the smoothness in the trajectory of the features in the image space is lost while the neuro-control is used. This rough behavior results from the heuristic contribution of the RL neuro-controller which modifies the trend in IBVS schemes to follow straight trajectories in the image space. However it was observed during real-time tests that this roughness had no major affect in the tracking task nor in the image processing which is usually disturbed by non-smooth movements on the robot’s last actuator. Fortunately the robustness and high noise tolerance of the visual processing algorithm can effectively handle more radical movements of the robot’s end-effector.

Figure 4(b) exhibits a more effective tracking with the camera attached to the end-effector following the straight trajectory in its way back to the target position of the features. However, as a consequence of the high initial visual error and the gain increment which follows, a considerable jump can be observed in the approaching zone of the step-response. This is later compensated in the final approach stage.

Notice that in simulation, it is possible to determine the lowest and maximum value of the trajectory generator gain with an acceptable regulation. The neuro-controller contribution can therefore be fixed to fall at this range in order to avoid stability problems derived from the on-line gain adjustment. This is only a practical solution but further analysis must be
conducted to define a set of valid stability criteria to assure robustness of the overall control structure.

New research ideas have suggested the construction of a new RL framework to include the analysis of the rate of change of the visual features on the image plane in order to generate new gain values to accordingly compensate for high oscillatory or striking trajectories in the visual features. A second idea is to decouple the gains in the trajectory generator for stronger robotic links like those on the robot’s base and for other lighter actuators like those usually conforming the robot’s end-effector. So a different gain set can be applied to less robust links and to those links demanding higher energy consumption. This may improve the adjustment policy applied by the actor network.

ACKNOWLEDGEMENTS

The authors would like to thank to the Chancellor’s office, to the Electronics and Computer Science Division and to the IT Technologies Office, all from the Exact Sciences and Engineering Campus CUCEI, in the University of Guadalajara, for providing us with funding support and access to their research facilities. Special thanks to Dr. Victor Gonzalez Alvarez for all his invaluable help to publish this paper and also thanks to Roy Moody, who has donated the MA2000 robot used in this research.

REFERENCES