A Computational Cognition and Visual Servoing based Methodology to Design Automatic Manipulative Tasks

Hendry Ferreira Chame and Philippe Martinet
Robotics Team of the Institut de Recherche en Communications et Cybernétique de Nantes (IRCCyN), Nantes, France

Keywords: Cognitive Robotics, Computational Cognition, Artificial Intelligence, Visual Servoing.

Abstract: In the last decades, robotics has exerted an important role in the research on diverse knowledge domains, such as, artificial intelligence, biology, neuroscience and psychology. In particular, the study of knowledge representation and thinking, has led to the proposal of cognitive architectures; capturing essential structures and processes of cognition and behavior. Robotists have also attempted to design automatic systems using these proposals. Though, certain difficulties have been reported for obtaining efficient low-level processing while sensing or controlling the robot. The main challenges involve the treatment of the differences between the computational paradigms employed by the cognitive and the robotic architectures. The objective of this work, is to propose a methodology for designing robotic systems capable of decision making and learning when executing manipulative tasks. The development of a system called the Cognitive Reaching Robot (CRR) will be reported. CRR combines the advantages of using a psychologically-oriented cognitive architecture, with efficient low-level behavior implementations through the visual servoing control technique.

1 INTRODUCTION

In the last decades, with the venue of fields of study such as cybernetics, artificial intelligence, neuroscience and psychology; remarkable progresses have been made in the understanding of what is required to create artificial life evolving in real-world environments (Arbib et al., 2008). Still, one of the remaining challenges is to create new cognitive models that would replicate high-level capabilities; such as, perception and information processing, reasoning, planning, learning, and adaptation to new situations.

The study of knowledge representation and thinking has led to the proposal of Cognitive Architectures. A Cognitive Architecture (CA) can be conceived as a broadly-scoped, domain-generic computational cognitive model, which captures essential structures and processes of the mind, to be used for a broad, multiple-level, multiple-domain analysis of cognition and behavior (Newell, 1994). For cognitive science (i.e., in relation to understanding the human mind) a CA provides a concrete mechaniscist framework for more detailed modeling of cognitive phenomena; through specifying essential structures, divisions of modules, relations between modules, and so on (Duch et al., 2008).

A robot that employs a CA to select its next action, is derived from integrated models of the cognition of humans or animals. Its control system is designed using that integrated CA and is structurally coupled to its underlying mechanisms (Sun, 2009). However, there are challenges associated with using these architectures in real environments; in particular, for performing efficient low-level processing (Hanford and Long, 2011). It can be hard, thus, to generate meaningful and trustful symbols from potentially noisy sensor measurements, or to exert control over actuators using the representation of knowledge employed by the CA.

Cognitive models are derived from a large spectrum of computational paradigms that are not necessarily compatible when considering underlying software architecture requirements. Scientists in cognition research, and actually higher-level robotic applications, develop their programs, models and experiments using a language grounded in an ontology based on general principles (Huelse and Hild, 2008). Hence, they expect reasonable and scalable performance for general domains and problem spaces.

On the side of cognitive robotists, it would not be reasonable to replace already existing robust mechanisms ensuring sensory-motor control by less efficient ones. Such is the case of the visual servoing technique which uses computer vision data to control the motion

Ferreira Chame H. and Martinet P..
A Computational Cognition and Visual Servoing based Methodology to Design Automatic Manipulative Tasks.
DOI: 10.5220/0004480802130220
In *Proceedings of the 10th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2013), pages 213-220
ISBN: 978-989-8565-70-9

of the robot's effector (Corke, 2011). This approach has the advantage of allowing the control by directly measuring the error on the effector's interaction with the environment; making it robust to inaccuracies in estimates of the system parameters (Chaumette and Hutchinson, 2006).

This research seeks to contribute to the debate standing from the point of view of cognitive roboticists. It can be conceived as an effort to assess to what extent it is feasible to build cognitive systems making use of the benefits of a psychologically-oriented CA; without leaving behind efficient control strategies such as visual servoing. The aim is to verify the potential benefits of creating an interactive platform under these technologies; and to analyze the resulting flexibility in automating manipulative tasks.

2 COGNITIVE ARCHITECTURES

According to (Kelley, 2003), two key design properties that underlie the development of any CA are *memory* and *learning*. Various types of *memory* serve as a repository for background knowledge about the world, the current episode, the activity, and oneself; while *learning* is the main process that shapes this knowledge. Based on these two features, different approaches can be gathered in three groups: *symbolic*, *non-symbolic*, and *hybrid* models.

A *symbolic* CA has the ability to input, output, store and alter symbolic entities; executing appropriate actions in order to reach goals (Newell, 1994). The majority of these architectures employ a centralized control over the information flow from sensory inputs, through memory; to motor outputs. This approach stresses the working memory executive functions, with an access to semantic memory; where knowledge generally has a *graph-based* representation. *Rule-based* representations of perceptions / actions in the procedural memory, embody the logical reasoning of human experts.

Inspired by connectionist ideas, a *sub-symbolic* CA is composed by a network of processing nodes (Duch et al., 2008). These nodes interact with each other in specific ways changing the internal state of the system. As a result, interesting emergent properties are revealed. There are two complementary approaches to memory organization, *globalist* and *localist*. In these architectures, the generalization of learned responses to novel stimuli is usually good, but learning new items may lead to problematic interference with existent knowledge (O'Reilly and Munakata, 2000).

A *hybrid* CA combines the relative strengths of the first two paradigms (Kelley, 2003). In this sense, *symbolic* systems are good approaches to process and executing high-level cognitive tasks; such as, planning and deliberative reasoning, resembling human expertise. But they are not the best approach to represent low-level information. *Sub-symbolic* systems are better suited for capturing the context-specificity and handling low-level information and uncertainties. Yet, their main shortcoming are difficulties for representing and handling higher-order cognitive tasks.

3 VISUAL SERVOING

The task in visual servoing (VS) is to use visual features, extracted from an image, to control the pose of the robot's end-effector in relation to a target. The camera may be carried by the end-effector (a configuration known by *eye-in-hand*) or fixed in (*eye-to-hand*) (Corke, 2011). The aim of all vision-based control schemes is to minimize an error $e(t)$, which is typically defined by

$$e(t) = s(m(t), a) - s^* \tag{1}$$

The vector $m(t)$ is a set of image measurements used to compute a vector of k visual features $s(m(t), a)$, based on a set of parameters a representing potential additional knowledge about the system (i.e., the camera intrinsic parameters, or a 3-D model of the target). The vector s^* contains the desired values of the features.

Depending on the characteristics of the task, a fixed goal can be considered where changes in s depend only on the camera's motion. A more general situation can also be modeled, where the target is moving and the resulting image depends both on the camera's and the target's motion. In any case, VS schemes mainly differ in the way s is designed. For image-based visual servo control (IBVS), s consists of a set of features that are immediately available in the image data. For position-based visual servo control (PBVS), s consists of a set of 3D parameters, which must be estimated from image measurements. Once s is selected, a velocity controller relating its time variation to the camera velocity is given by

$$\dot{s} = L_s V_c \tag{2}$$

The spatial velocity of the camera is denoted by $V_c = (v_c, \omega_c)$, with v_c the instantaneous linear velocity of the origin of the camera frame and ω_c the instantaneous angular velocity of the camera frame. $L_s \in R^{6\times k}$ is named the interaction matrix related to s.

Using (1) and (2), the relation between the camera velocity and the time variation of e can be defined by

$$\dot{e} = L_e V_c \tag{3}$$

Considering V_c as the input to the controller, if an exponential decoupled decrease of e is desired, from (3) the velocity of the camera can be expressed by

$$V_c = -\lambda L^+{}_e e \tag{4}$$

where $L^+ \in R^{6\times k}$ is chosen as the Moore-Penrose pseudoinverse of L_e, that is $L^+{}_e = (L_e{}^t L_e)^{-1} L_e{}^t$ when L_e is of full rank 6. In case $k = 6$ and $det(L_e) \neq 0$, it is possible to invert L_e giving the control $V_c = -\lambda L_e{}^{-1} e$.

Following (4), the six components of V_c are given as input to the controller. For robots with less than six degrees of freedom, the control scheme may be expressed in the joint space by

$$\dot{q} = -\lambda(J_e{}^+ e + P_e e_s) - J_e{}^+ \frac{\partial e}{\partial t} \tag{5}$$

where J_e is the feature Jacobian matrix associated with the primary task e, $P_e = (I_6 - \widehat{J}_e{}^+ \widehat{J}_e)$ is the gradient projection on the null space of the primary task to accomplish a secondary task e_s, and $\frac{\widehat{\partial e}}{\partial t}$ models the motion of the target. An example of VS is presented in Figure 1.

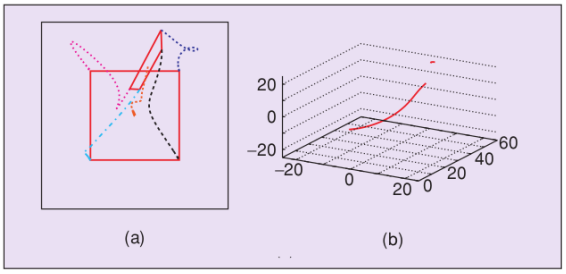


Figure 1: IBVS example. (a) Image points trajectories (c) 3-D trajectory of the camera optical center (Chaumette and Hutchinson, 2007).

4 THE CRR PROPOSAL

The Cognitive Reaching Robot (CRR) is a system designed to perform interactive manipulative tasks. When compared to non-cognitive approaches, CRR has the advantage of being adaptive to variations of the task; since the reinforcement learning mechanism reduces the need for explicitly reprogramming the behavior of the robot. Furthermore, CRR is robust to changes in the robotic system due to wear. It is tolerant to calibration errors by employing visual servoing; where modeling errors are compensated in the control loop (the camera directly measures the task errors).

The platform presents a modular organization (as shown in Figure 2) and is composed by three modules. The cognitive module is responsible for symbolic decision making and learning. The auditory module processes speech recognition. The visuomotor module is in charge of applying the VS control. To enable inter-modular communication, six topics were defined. Topics are named buses over which modules exchange messages. According to the sensory modalities that compose CRR, auditory, proprioceptive and visual topics were defined. The aim of these topics is sending sensory information to the cognitive module. Similarly, the cognitive module sends commands to the auditory, visual and proprioceptive modules.

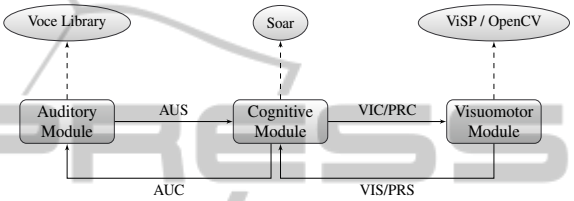


Figure 2: The CRR architecture. The boxes represent modules and the ovals indicate the libraries wrapped inside the modules. The links between modules indicate topics. AUS: auditory sensory, PRS: proprioceptive sensory, VIS: visual sensory, AUC: Auditory command, VIC: Visual command, PRC: Proprioceptive command.

Hardware Components. The design of CRR aimed to praise the reusability of equipments, so its hardware components were chosen according to a criteria of accessibility in the robotic lab. The project considered a Stäubli TX-40 robot manipulator, an AVT MARLIN F-131C camera, and a DELL Vostro 1500 laptop (Intel Core 2 Duo 1.8GHz 800Mhz FSB, 4.0GB DDR2 667MHz RAM, 256MB NVIDIA GeForce 8600M GT).

Software Components. Three criteria grounded the choice for software technologies: source availability, efficiency and continuity of the development community. The sole exception was the use of SYMORO+ (Khalil and Creusot, 1997), a proprietary automatic symbolic modeling tool for robots. CRR was developed under Ubuntu Oneiric Ocelot and relied on the Voce Library 0.9.1, ViSP 2.6.2, the *symbolic* CA Soar 9.3.2, and ROS Electric. Eclipse Juno 4.2 was used for testing the algorithms.

5 CASE STUDY

The experimental situation designed, consisted in a reaching, grasping, and releasing task, involving reinforcement learning. From the inputs received, and based on the rewards or punishments obtained, the

robot must learn the optimal sequence policy $\pi : S \rightarrow A$ to execute the task, and thus, to maximize the reward obtained.

Task Definition. The experimenter is positioned in front of the robot for every trial and presents it an object accompanied by a verbal auditory cue (”wait” or ”go”). The robot has to choose between sleeping or reaching the object. If the object is reached after a ”wait” or the robot goes sleeping after a ”go”, the experimenter sends an auditory verbal cue representing punishment (”stop”) and the trial ends. On the contrary, if the robot goes sleeping after getting a ”wait”or follows the object after a ”go”, it receives an auditory verbal cue representing reward (”great”). After being rewarded for following the object, the experiment enters the releasing phase. If the robot alternated the location for dropping the object it is rewarded, otherwise it is punished. Figure 3 presents the reinforcement algorithm.

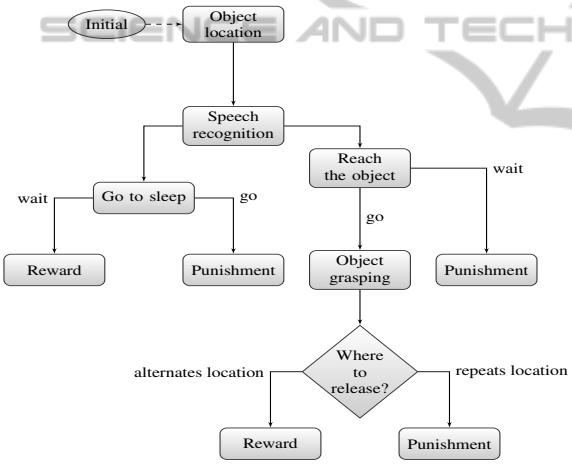


Figure 3: Task reinforcement algorithm.

The robot has two main goals in the experiment. It is required to learn when reaching or sleeping in the presence of the object; and if the object is grasped, to learn to drop it alternatively in one of two containers. Summarizing, the robot is required of perceptive abilities (recognizing the object and speech), visuomotor coordination, and decision making (while remembering events).

5.1 Perception

Object Recognition. The recognition of the object was accomplished using OpenCV 2.4. The partition of the image into meaningful regions was achievement in two steps. The classification steps includes a decision process applied to each pixel assigning it to one of C classes $C \in \{0...C-1\}$. For CRR a particular case using $C = 2$ known as *binarization* (Pratt, 2007) was used. Formally, it is conceived as a monadic operation taking an image of size $I^{W \times H}$ as input, and producing an image $O^{W \times H}$ as output; such as

$$O[u,v] = f(I[u,v]), \forall (u,v) \in I \tag{6}$$

The color image I is processed in HSV color space, and the f function used was

$$f(I[u,v]) = \begin{cases} 1 & \text{if } \varepsilon_i < I[u,v] < \varepsilon_f \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

The choice of f was based on simplicity and ease of implementation; however, it assumes constant illumination conditions throughout the experiment (which is the case since the environment is illuminated artificially). The thresholds ε were set to recognize red objects.

In the description phase the represented sets S are characterized in terms of scalar or vector-valued features such as size, location and shape. A particularly useful class of image features are moments (Corke, 2011), which are easy to compute and can be used to find the location of an object (centroid). For a binary image $B[x,y]$ the $(p+q)^{th}$ order moment is defined by

$$m_{pq} = \sum_{y=0}^{y_{max}} \sum_{x=0}^{x_{max}} x^p y^q B(x,y) \tag{8}$$

Moments can be given a physical interpretation by regarding the image function as a mass distribution. Thus m_{00} is the total mass of the region, and the centroid of the region is given by

$$x_c = \frac{m_{10}}{m_{00}}, y_c = \frac{m_{01}}{m_{00}} \tag{9}$$

After the centroid is obtained, the last step consisted in proportionally defining two points beside it, forming an imaginary line of -45° slope. These two points are the output of the object recognition algorithm, later entered to ViSP to define 2D features and performing the VS control.

Speech Recognition. CCR used the Voce Library 0.9.1 to process speech. It required no additional efforts than changing the grammar configuration file to include the vocabulary to be recognized.

5.2 Visuomotor Control

In order to perform visuomotor coordination to reach the object, an IBVS strategy was chosen given its robustness to modeling uncertainties (Chaumette and Hutchinson, 2006). The camera was located in the

effector of the robot (*eye-in-hand*), thus the J_e component of (5) is defined by

$$J_e = L_e{}^c V_n{}^n J(q) \tag{10}$$

Two visuomotor subtasks were defined: reaching the object and avoiding joint limits.

Primary Task. The subtask e consisted in positioning the end-effector in front of the object for grasping it. The final orientation of the effector was not important (assuming a spherical object), therefore, only 3 DOF were required to perform the task. Two 2D point features were used given its simplicity, each of them allowing to control 2 DOF. The resulting interaction matrix L_{e_i} was defined by

$$L_{e_i} = \begin{bmatrix} -1/Z_e & 0 & x_e/Z_e & x_e y_e & -(1+x_e^2) & y_e \\ 0 & -1/Z_e & y_e/Z_e & (1+y_e^2) & -x_e y_e & -x_e \end{bmatrix} \tag{11}$$

the error vector for the primary task can be expressed by

$$e^t = \begin{bmatrix} (x_{s_1} - x_{s_1{}^*}) & (y_{s_1} - y_{s_1{}^*}) & (x_{s_2} - x_{s_2{}^*}) & (y_{s_2} - y_{s_2{}^*}) \end{bmatrix} \tag{12}$$

and $L_e^{4\times 6}$ is given by

$$L_e = \begin{bmatrix} L_{e_1} & L_{e_2} \end{bmatrix}^t \tag{13}$$

Secondary Task. The remaining 3 DOF were used to perform the secondary task of avoiding joint limits. The strategy adopted was *activation thresholds* (Marchand et al., 1996). The secondary task is required only if one (or several) joint is in the vicinity of a joint limit. Thus, thresholds can be defined by

$$\begin{aligned} \widetilde{q}_{i_{min}} &= q_{i_{min}} + \rho(q_{i_{max}} - q_{i_{min}}) \\ \widetilde{q}_{i_{max}} &= q_{i_{max}} - \rho(q_{i_{max}} - q_{i_{min}}) \end{aligned} \tag{14}$$

where $0 < \rho < 1/2$. The vector e_s had 6 components, each defined by

$$e_{s_i} = \begin{cases} \frac{\beta(q_i - \widetilde{q}_{i_{max}})}{q_{i_{max}} - q_{i_{min}}} & \text{if } q_i > \widetilde{q}_{i_{max}} \\ \frac{\beta(q_i - \widetilde{q}_{i_{min}})}{q_{i_{max}} - q_{i_{min}}} & \text{if } q_i < \widetilde{q}_{i_{min}} \\ 0 & else \end{cases} \tag{15}$$

with the scalar constant β regulating the amplitude of the control law due to the secondary task.

5.3 Decision Making

Markov Decision Process (MDP) provided the mathematical framework for modeling decision making. The task space was represented by a set of $S = \{S_0, ..., S_{10}\}$ states, $A = \{a_0, ..., a_8\}$ actions and $Pa(s,s') = \{\alpha_0, ..., \alpha_{14}\}$ action-transition probabilities. The simplified MDP representation of the agent is given in Figure 4.

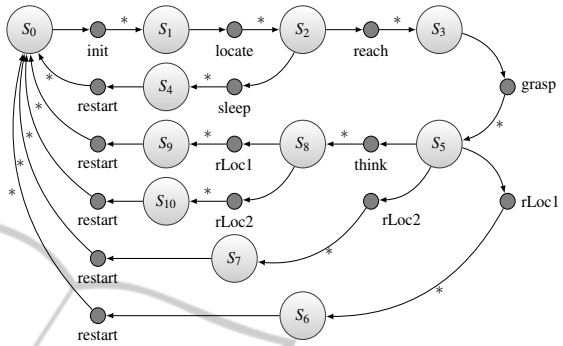


Figure 4: The MDP task model. $* = (\alpha, \rho)$, where α is the transition probability from s to s' when taking the *action*, and ρ is the reward associated with the *state*. From all actions there is a link to S_0 (omitted for clarity) modeling errors on the process with probability $1 - \alpha$. The states are: S_0: Started, S_1: Initialized, S_2: Object located, S_3: Object reached, S_4: Sleeping, S_5: Object grasped, S_6: Object released in location 1, S_7: Object released in location 2, S_8: Thinking, S_9: Object released in location 1 after thinking, S_{10}: Object released in location 2 after thinking. The action a_0 initializes the system, a_1 signals the localization of the object, a_2 signals the robot to reach the object, a_3 puts the robot in sleeping mode, a_4 signals the robot to close the gripper, a_5 explores past events, a_6 and a_7 signal the robot to release the object at location 1 or 2 respectively, and a_8 restarts the system. If a state receives a negative feedback from the user $\rho_i = -4$ (punishment). In case of positive feedback, $\rho_i = 2$ (reward).

Procedural Knowledge Modeling. Cognitive models in Soar 9.3.2 are stored in long-term production memory as productions. A production has a set of conditions and actions. If the conditions match the current state of working memory (WM), the production fires and the actions are performed. Some attributes of the state are defined by Soar (i.e., *io*, *input-link* and *name*) ensuring the operation of the architecture. The modeler has the choice to define custom attributes, which derives in a great control over the state.

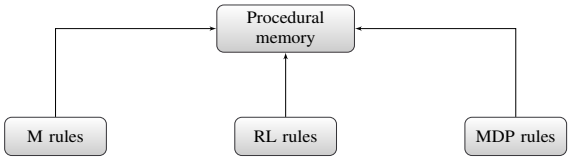


Figure 5: Procedural memory. M: Maintainance, RL: Reinforcement Learning, MDP: Markof Decision Process.

The procedural knowledge implementation in Soar can be conceived as a mapping between an in-

put to an output semantic structure. To develop the case study, it was necessary to define three types of productions: maintenance, action and learning rules. The first category includes rules that process inputs and outputs to maintain a consistent state in WM; a typical task is clearing or putting data into the slots in order to access the modules functionalities. The second category includes rules related to the robot's task, such as, managing the MDP state transitions. The last group involves rules that guarantee the correct functioning of RL; it includes tasks like maintaining the operators' Q-values, or registering rewards and punishments. Figure 5 presents a qualitative view of the contents of the procedural memory. For modeling the case study, a total of 57 productions were defined.

Remembrance of Events. Functionalities in Soar are accessed through testing the current semantic structure of WM. The same principle applies for querying data in the long term memory. In order to access the episodic or semantic memory, the programmer must define rules placing the query attributes and values on the attribute *epmem* (for episodic retrieval) or *smem* (for semantic retrieval). After each decision cycle, Soar checks the *epmem.command* node to match conditions for episodic retrieval. A copy of the most recent match (if found) will be available on the *epmem.result* for the next decision cycle.

Remembrance of Facts. Facts about the world can be modeled through semantic structures. For the case study, the agent must know what are the stimuli received, or at least, how it feels like in relation to them. Thus, semantic information concerning stimuli was added to the system. The resulting graph was equivalent to a tree of height two (Figure 6). A stimulus has a name, a sensory modality (visual, auditory or proprioceptive) and a valence (positive, negative or neutral).

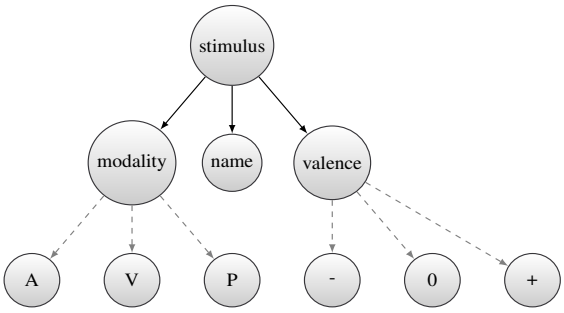


Figure 6: Stimulus semantic knowledge. *A*: Auditory, *V*: Visual, *P*: Proprioceptive.

Reinforcement Learning. The learning by reinforcement can be considered as equivalent to mapping situations to actions, so as to maximize a numerical reward signal (Kaelbling et al., 1996). The learner is not told which actions to take, but instead it must discover which actions yield the most reward by trying them. The RL module of Soar is based on the Q-learning algorithm (Kaelbling et al., 1996). In the case study a reward is applied whenever the state is not neutral. Figure 7 illustrates the processing of the stimuli. When an input arrives, procedural rules query the semantic memory to determine the valence associated with the stimulus. Following an analogy with respect to humans, the agent continues to work if it doesn't feel happy or sad about what it has done; if so, it stops to think about it.

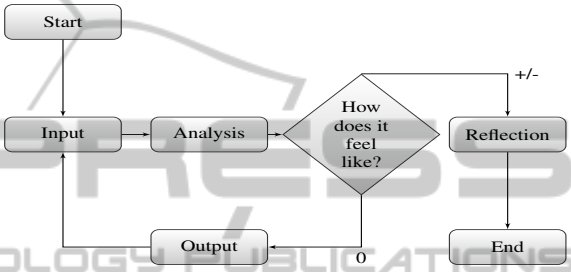


Figure 7: Stimulus processing and reinforcement.

6 RESULTS

The implementation of the functionalities of CRR took place incrementally. Given the independence between the different modules, each component could be developed and tested individually. The modules were connected to the platform through ROS Electric; a comprehensive simulation was done, and the results obtained are presented below.

System Performance. The performance of the visuomotor module is quite acceptable for real-time control applications. The module was designed to operate in four different modalities. In the *VS* mode, only visual servoing is available. In the *VSI* mode, it is possible to have a real-time view of the camera. In the *VSL* mode, the system generates log files for joint positions and velocities, feature errors, and camera velocities. Finally, a combination of the last three is allowed in the *VSIL* mode. As it can be seen in Figure 8, a freq. near to 66 Hz (approx. 15 ms per iteration) can be reached. If the camera view is displayed (which can be useful for debugging but has no importance for execution) the freq. drops to 20 Hz.

Joint Limit Avoidance. In order to test the joint limit avoidance property of the system, a simple simulation was designed. The robot was positioned in the configuration displayed in Figure 9a. An object is assumed to be presented to the robot, rotated -10° in

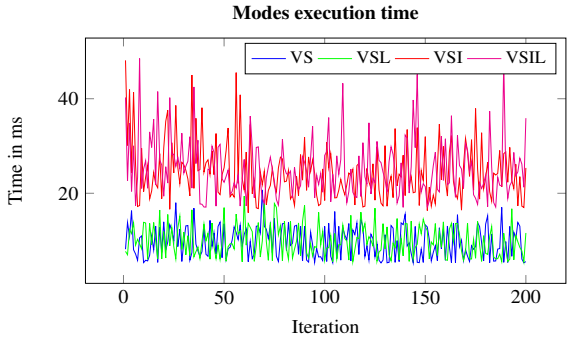


Figure 8: Visuomotor module computing time.

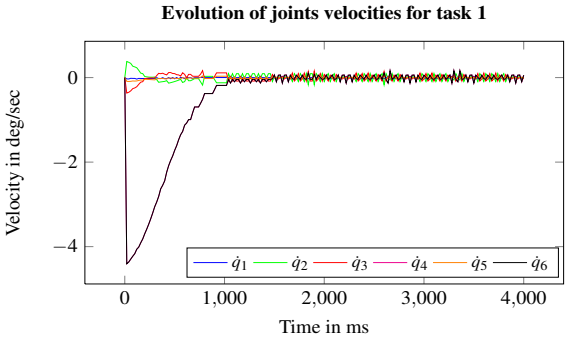
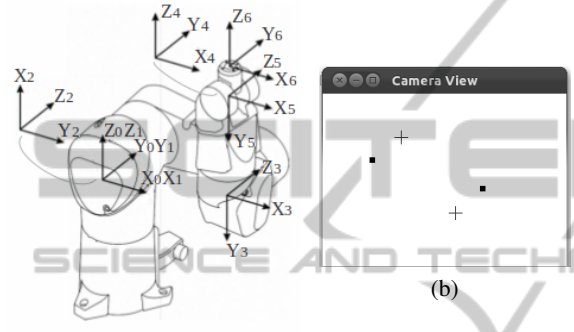


Figure 10: Simulation of VS primary task.



(a) (b)

Figure 9: Robot configuration for testing joint limits avoidance. a) Joint positions in deg: $q_1 = 0$, $q_2 = 90$, $q_3 = -90$, $q_4 = 0$, $q_5 = 0$, $q_6 = 0$. b) Simulated view, dots are the current feature locations and crosses are the desired locations.

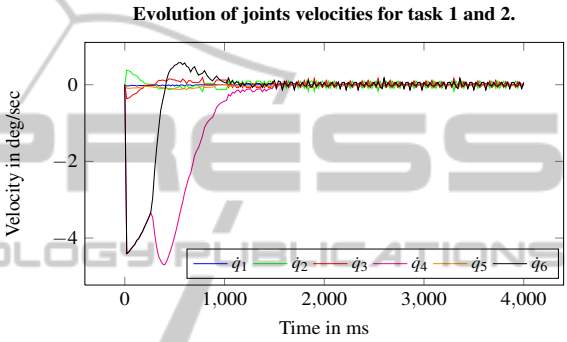


Figure 11: Simulation of VS avoiding joint limits.

the z axis of the camera frame. The simulated camera view is shown in Figure 9b.

The primary task (moving the robot to the desired view of the features) can be solved in infinite ways given the current singularity between joint frames 4 and 6. For testing the limit avoidance control law, limits of $q_{6_{min}} = -5°$ and $q_{6_{max}} = 5°$ were set to joint 6. As it is shown in Figure 10, if just the primary task is performed, the control law generated will mostly operate q_6 and the task will fall in local minima, since $q_{6_{min}}$ will be reached. On the contrary, as shown in Figure 11, setting a threshold $\rho = 0.5$ (which means it will be active when $q_6 < -2.5°$ or $q_6 > 2.5°$) solves the problem and the joint limit is avoided.

Task Learning. The task designed to run over CRR had two learning phases. In order to assess the correctness of the cognitive model and the learning algorithm; two experimental sets were defined. In the experimental set one (ES1), the objective was to teach the robot to identify when reaching the target. The ES1 evaluation consisted of five test cases varying the order of presentation of the clues "wait" and "go". In all conditions the robot started without prior knowledge (the RL module was reset). The comparison between a RL and a random police is given in Table 1; as it can be seen, the robot was able to learn the task. The experimental set two (ES2) assumes ES1 was accomplished, so the agent properly grasped the object and must now learn where to drop it. The ES2 evaluation showed the agent was able to quickly learn the task using RL, and the resulting Q-values are presented in Table 2. For each test case of both ES1 and ES2, the first 20 responses of the robot were registered.

Table 1: ES1 evaluation results. RL-S: number of successes applying a RL policy, RL-C: RL-S/attempts, R-S: number of successes applying a random policy, R-C: R-S/attempts.

Test	RL-S	RL-C	R-S	R-C
C1	17	0.85	8	0.40
C2	18	0.90	11	0.55
C3	17	0.85	12	0.60
C4	18	0.90	9	0.45
C5	18	0.90	10	0.50

7 CONCLUSIONS

This work started from the interest in developing cognitive robotic systems for executing manipulative

Table 2: ES2 evaluation results. The robot attempted to release the object without remembering 5 times (taking the *release-loc-1* and *release-loc-2* actions). However, it learned to maximize the reward by tacking the *think-Remember* action, which was selected 15 times. Finally, after recalling the last location, the agent learned to alternate between the *think-release-loc-2-B* and *think-release-loc-1-A* actions.

Action	Freq.	Reward
think-Remember	15	4.9302
think-release-loc-2-A	1	-2.2800
think-release-loc-2-B	7	6.9741
think-release-loc-1-A	7	6.9741
think-release-loc-1-B	0	0.0000
release-loc-2	2	0.6840
release-loc-1	3	0.4332

tasks. To this purpose, an approach emphasizing multidisciplinary theoretical and technical formulations was adopted. A methodological proposal for integrating a psychologically-oriented cognitive architecture to the visual servoing control technique has been presented; and resulted in the development of a modular system capable of auditory and visual perception, decision making, learning and visuomotor coordination. The evaluation of the case study, showed that CRR is a system whose operation is adequate for real-time interactive manipulative applications.

ACKNOWLEDGEMENTS

This research was accomplished thanks to the founding of the National Agency of Research through the EQUIPEX ROBOTEX project (ANR-10-EQX-44), of the European Union through the FEDER ROBOTEX project 2011-2015, and of the Ecole Centrale of Nantes.

REFERENCES

(2008). *Proceedings of the IROS workshop on Current software frameworks in cognitive robotics integrating different computational paradigms*. 22 September, Nice, France.

Arbib, M. A., Metta, G., and van der Smagt, P. P. (2008). Neurorobotics: From vision to action. In *Springer Handbook of Robotics*, pages 1453–1480.

Chaumette, F. and Hutchinson, S. (2006). Visual servo control, part i: Basic approaches. *IEEE Robotics and Automation Magazine*, 13:82–90.

Chaumette, F. and Hutchinson, S. (2007). Visual servo control, part ii: Advanced approaches. *IEEE Robotics and Automation Magazine*, 14(1):109–118.

Corke, P. I. (2011). *Robotics, Vision & Control: Fundamental Algorithms in Matlab*. Springer.

Duch, W., Oentaryo, R. J., and Pasquier, M. (2008). Cognitive architectures: Where do we go from here? In (iro, 2008), pages 122–136. 22 September, Nice, France.

Hanford, S. and Long, L. (2011). *A cognitive robotic system based on the Soar cognitive architecture for mobile robot navigation, search, and mapping missions*. PhD thesis, Aerospace Engineering, University Park, Pa., USA.

Huelse, M. and Hild, M. (2008). A brief introduction to current software frameworks in cognitive robotics integrating different computational paradigms. In (iro, 2008). 22 September, Nice, France.

Kaelbling, L., Littman, M., and Moore, A. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Kelley, T. D. (2003). Symbolic and sub-symbolic representations in computational models of human cognition: What can be learned from biology? *Theory & Psychology*, 13(6):847–860.

Khalil, W. and Creusot, D. (1997). Symoro+: A system for the symbolic modelling of robots. *Robotica*, 15(2):153–161.

Marchand, E., Chaumette, F., and Rizzo, A. (1996). Using the task function approach to avoid robot joint limits and kinematic singularities in visual servoing. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS'96*, volume 3, pages 1083–1090, Osaka, Japan.

Newell, A. (1994). *Unified Theories of Cognition*. William James Lectures. Harvard University Press.

O'Reilly, R. and Munakata, Y. (2000). *Computational Explorations in Cognitive Neuroscience: Understanding the Mind by Simulating the Brain*. Bradford Books. Mit Press.

Pratt, W. (2007). *Digital Image Processing: PIKS Scientific Inside*. Wiley-Interscience publication. Wiley.

Sun, R. (2009). Multi-agent systems for society. chapter Cognitive Architectures and Multi-agent Social Simulation, pages 7–21. Springer-Verlag, Berlin, Heidelberg.