Task Sequencing for High Level Sensor-Based Control

Nicolas Mansard, François Chaumette

***Abstract*— Classical sensor-based approaches tend to constrain all the degrees of freedom of a robot during the execution of a task. In this article a new solution is proposed. The key idea is to divide the global full-constraining task into several subtasks that can be applied or inactivated to take into account potential constraints of the environment. Far from any constraint, the robot moves according to the full task. When it comes closer to a configuration to avoid, a higher level controller removes one or several subtasks, and activates them again when the constraint is avoided. The last controller ensures the convergence at the global level by introducing some look-ahead capabilities when a local minimum is reached. The robot accomplishes the global task by automatically sequencing sensor-based tasks, obstacle avoidance and short deliberative phases. In this article, a complete solution to implement this idea is proposed, along with several experiments that prove the validity of this approach.**

***Index Terms*— Sensor-based control, tasks sequencing, redundancy, avoidance, planning, visual servoing**

I. Introduction

SENSOR-feedback control loop techniques, such as visual servoing [15], [11] provide very efficient solutions to control robot motions. It supplies high positioning accuracy, good robustness to sensor noise and calibration uncertainties, and reactivity to environment changes. However, the convergence domain is often local: if the initial error is large, such a control may become erratic or even impossible [5]. By adequately choosing the sensor features used for the control, like in 2-1/2-D visual servoing [24] or by using image moments [40], the convergence domain is enlarged and the robot behavior is enhanced without loosing the good properties of accuracy and robustness. However these solutions are inefficient in taking environment constraints into account. Such constraints are generally considered as a secondary task [21], [28]. In that case they can not be completed if the main task involves all the robot degrees of freedom (DOF). A second solution is to realize a trade off between the main task and the constraints [30] but with no guarantee about control convergence or constraints being respected.

A vast number of trajectories are generally available to reach the goal. The classical control schemes choose a particular trajectory without knowing if it is valid or not. In certain cases, this trajectory may lead to instability or singularity. Reactive avoidance methods such as [21], [17], [30] simply modify this trajectory locally, which is not always sufficient.

To always obtain an optimal execution, higher level control chooses in advance the optimal trajectory by planning a path to be followed, for example in the sensor space [9], [29]. This provides a complete solution, which ensures optimality, stability and physical feasibility to the goal when it is reachable. It is also able to take several environment constraints into account, ensuring for example that the tracked object remains in the camera field of view or the robot avoids its joint limits. Path planning solves the deficiency of the low-level methods but it is consequently hardly reactive to environment changes or execution errors such as localization uncertainties. Some methods have been proposed to reactively modify the path [33], [19]. But these methods provide only a local convergence of the modified path, and still require a lot of knowledge about the environment to compute the initial path.

Several works have tried to take advantage of these two solutions, generally by modifying the low-level control loop with respect to a higher controller level. One approach is to sequence several simple tasks using an *a priori* order [31], [38], [32]. This provides a good robot behavior, but the choice of the tasks to be sequenced along with the order have to be tuned by hand for each application. A second set of solutions are the switching systems: rather than deciding in advance which path or which task should be used to reach the goal, switched systems use a set of subsystems along with a discrete switching control [12], [7]. The robot then avoids difficult regions by switching from a first control law (a particular trajectory) to another one when necessary. This enlarges the stable area to the union of the stable area of each task used. A last solution is to divide the global task into several subtasks that are activated or inactivated according to the current environment state. In [3] a mobile manipulator moves according to two subtasks. The first one with higher priority is a deformable-path following. The second one is a positioning of the embedded arm into the fixed world frame, that compensates the motions of the mobile platform. When the deformable path is too far from the initially planned path, the second subtask has to be suspended because it is impossible to achieve. A similar idea was used in [41] to control a highly redundant humanoid robot. The robot moves in the Cartesian plane using a simple three-dimensional task. A new subtask is activated to take obstacles into account only when this task fails. However, the controller and the criteria proposed in these works to suspend and to activate the secondary subtasks are difficult to generalize to other platforms.

Manuscript submitted November 20, 2005; revised June 1, 2006; accepted for publication September 10, 2006. This paper was presented in part at the IEEE International Conference on Robotics and Automation, Barcelona, Spain, April 2005.

The authors are with IRISA/INRIA Rennes, Lagadic Project, Campus de Beaulieu, 35042 Rennes-cedex, France. E-mail: {nmansard, chaumett}@irisa.fr.

In this work, a general method is proposed to sequence tasks to reach the goal taking reactively into account several environment constraints. The key idea is to separate a complete servoing task into several subtasks, that use only a subspace of all the robot DOF. At each step, the robot moves to achieve the active subtasks, until it reaches the goal position where all the subtasks are applied and realized. A higher-level controller can remove or put back some subtasks, in order to relax some DOF. These available DOF are used to take into account additional constraints such as visual-occlusion or joint-limit avoidance. More precisely when the robot comes close to violate a constraint, the higher-level controller chooses the adequate DOF to be used to ensure the constraint is respected, and removes the corresponding subtask from the active ones. This subtask is later put back, when the robot no more violates the constraint.

This paper presents a complete method to realize this general idea. The complete controller is composed of several layers that provide a good robot behavior at all levels, from a local and accurate convergence to the convergence from a very distant initial position around obstacles. To provide a good overview on this scheme, the global structure including all the controllers is first presented in Section II. The complete system is then built bottom-up, each different layer being detailed in a different section from Section III to V. The described method is general and can be applied for all sensor-feedback control methods. For this article, it was nevertheless applied to visual servoing. The additional constraints are classical avoidances that can be encountered in real robotic system, such as joint-limit, visual-occlusion and obstacle avoidance. The subtasks and constraints used to realize the experiments are briefly presented in Section VI. The experimental results are finally set out in Section VII.

II. Controller architecture

We first present the global architecture of the system to provide a large overview of the controllers detailed in the next sections. The system is composed of four layers of controllers, each stage controlling the actions of the controllers above it. Figure 1 sums up the architecture.

1) The first controller is composed of a stack which orders the subtasks currently active. Only the subtasks in the stack are taken into account in the control law. The subtask at the bottom level has priority over all the others, and the priority decreases as the stack level increases. The control law is computed from the subtasks in the stack, in accordance with three rules:

- any new subtask added in the stack does not disturb the subtasks already in the stack.
- the control law is continuous, even when a subtask is added or removed from the stack. The robot is controlled through the articular velocity $\dot{\mathbf{q}}$. A break of continuity would mean an infinite acceleration during a short period of time, which would imply that the control is not correctly applied.
- if possible, the additional constraints should be added to the control law, but without disturbing the subtasks in the stack.

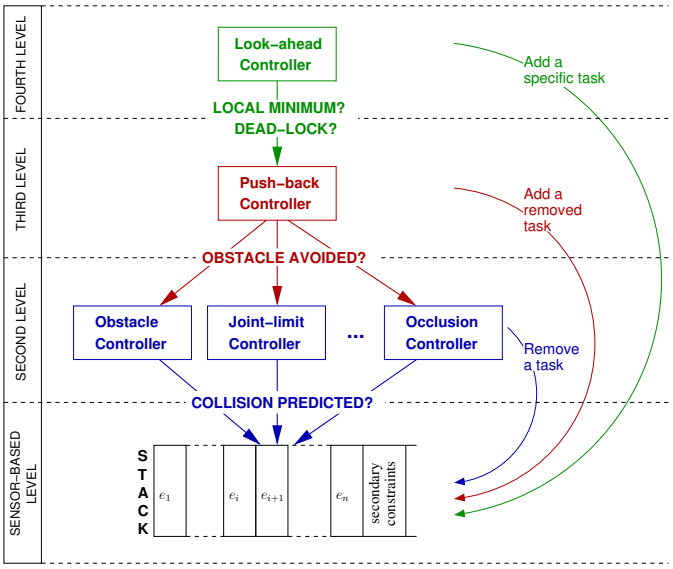


Fig. 1. Architecture of the global system, composed of four controller layers. The first low level (sensor-based level) computes the control law from the stack. At the second level, a first set of controllers (in blue in the figure) ensures that the environment constraints are respected by removing subtasks when needed. Upper level (push-back controller, in red in the figure) pushes the removed subtasks back in the stack when the corresponding constraint is satisfied. Finally, the top level (convergence controller, in green in the figure) ensures the convergence of the bottom controllers by ending potential local minimum and dead-lock.

The control law is computed from the stack, using the redundancy formalism introduced in [35], [37]. The additional constraints are added at the very top of the stack, which means that they are taken into account only if some DOF remain free after applying the active subtasks. This priority order may seem illogical, considering that the constraints are obstacles that the robot should avoid above all. However, the positioning task has priority since it is the task we want to see completed, despite the presence of the obstacles. The second level controller is then used to ensure that the constraints are respected when it is obvious that the robot will violate them.

2) The second controller ensures that enough DOF remain free to take the constraints into account, and thus that the environment constraints are respected. The controller detects that the constraints are not sufficiently taken into account by a linear prediction over the robot position with respect to the applied control law. When a constraint violation is predicted for the next few iterations, the controller selects the optimal subtask to be removed from the stack using the measures we proposed in [26]. In some cases, the removing of one subtask is not enough to satisfy the constraint. This could occur when the dynamic of the constraint is too high with respect to the robot reactivity, or when the necessary DOF is shared by two tasks that are both to be removed before the constraint is properly taken into account. In this case, the controller removes a second task at next iteration, and so on.

3) The third controller observes the subtasks that have been removed from the stack by the second controller and try to put them back in the stack as soon as possible. At the beginning of the servo, all the subtasks are in the stack. A subtask outside the stack can thus always be linked to a constraint that was the

reason of its removal. The controller computes the effects on the control law due to the reinsertion of the removed subtask into stack. The subtask is put back in the stack when no constraint violation is predicted any more.

4) The top controller ensures the convergence of the system by solving the dead locks of the bottom controllers. The three bottom controllers ensure only a local convergence. Two problems may occur while using only these controllers:

- at some moment which will be emphasized in Section V-A, the third controller may be unable to put back a removed subtask in the stack. One subtask is thus in a *local minimum*. This is easily detectable: a local minimum occurs when all the subtasks in the stack are completed while a subtask remains out of the stack.
- on the contrary it may happen that the third controller puts back a subtask too early. The subtask will then be removed some times later for the same reason as before, then put back again. These *dead-locks* can be detected by looking at a loop in the execution graph.

In these cases the controller adds a new specific task into the stack that is dedicated to solve these problems, for example by specifying an intermediary goal to reach or by computing a local path to follow. The corresponding mechanism is detailed in Section V-B.

The differences between our strategy and classical path planning are thus significant. With the control strategy proposed above, the robot is able to reach the goal by using only the low level sensor-based controllers in the general case. Only the current available sensor values are thus needed at each iteration. In very difficult situations the low-level minimization-based control is not sufficient. The last controller then gets the robot out from the local minimum by using some global knowledge such as a map or some *a-priori* about the robotic system setup. In this case, the last controller is not used till the end of the servo but only until the local minimum is left. Using this global scheme, the robot execution keeps the good properties of sensor-based control (rapidity, accuracy, low computation rates...) along with a large convergence domain provided by the look-ahead capabilities of the top controller.

III. Sensor-based control using a stack of tasks

In this section, the control law of the first controller is designed. This controller is based on a stack of tasks, composed of the current active tasks, and on the constraints which have to be taken into account. This stack makes possible very simple actions on the robot, such as activate a task (put a task in the stack), remove a task or swap the priority between two tasks.

We explain first how to sequence tasks and to maintain the tasks already achieved. Section III-A recalls the redundancy formalism [21], [14]. It has first been used for sensor-based control in [35] and in numerous applications since (e.g. visual servoing in [11], force distribution for the legs of a walking machine [18], or human-machine cooperation using vision control [13]). The idea is to use the DOF left by a first task to realize a secondary task at best without disturbing the first one. The major advantage of the redundancy formalism with respect to other methods that join two objectives in one control law (such as [30] and [4]) is that the secondary task has no effect on the task having priority due to the choice of an appropriate projection operator.

Section III-B sets out the way the redundancy formalism is used to stack several subtasks. The method presented here has been first proposed in [14] and formalized in [37]. It has often been used since for highly redundant systems such as humanoids [36] or virtual-entity control for animation [1]. In Section III-C, we briefly recall the method proposed in [25] to ensure the control law continuity, using a non homogeneous first order differential equation. Finally, the Gradient Projection Method (GPM) is recalled in Section III-D. This method has been first proposed for non-linear optimization [34]. It has been widely used for dealing with various types of constraints in robotic (see for example [21], [22] for joint-limit and singularity avoidance, [17] for obstacle avoidance or [28] for occlusion avoidance). The final control law used is given in Section III-E.

A. Redundancy formalism for two tasks

Let $\mathbf{q}$ be the articular vector of the robot. Let $\mathbf{e_1}$ and $\mathbf{e_2}$ be two tasks, $\mathbf{J_i} = \frac{\partial \mathbf{e_i}}{\partial \mathbf{q}}$ $(i = 1, 2)$ their Jacobian, defined by:

$$\dot{\mathbf{e}}_\mathbf{i} = \frac{\partial \mathbf{e_i}}{\partial \mathbf{q}} \dot{\mathbf{q}} = \mathbf{J_i} \dot{\mathbf{q}} \tag{1}$$

Since the robot is controlled using its articular velocity $\dot{\mathbf{q}}$, (1) has to be inverted. The general solution (with $i = 1$) is:

$$\dot{\mathbf{q}} = \mathbf{J_1^+} \dot{\mathbf{e}}_\mathbf{1} + \mathbf{P_1 z} \tag{2}$$

where $\mathbf{P_1}$ is the orthogonal projection operator on the null space of $\mathbf{J_1}$ and $\mathbf{J_1^+}$ the pseudoinverse of $\mathbf{J_1}$. Vector $\mathbf{z}$ can be used to apply a secondary command, that will not disturb the task $\mathbf{e_1}$ having priority. Here, $\mathbf{z}$ is used to carry out at best the task $\mathbf{e_2}$. Introducing (2) in (1) (with $i = 2$), we obtain:

$$\dot{\mathbf{e}}_\mathbf{2} = \mathbf{J_2 J_1^+} \dot{\mathbf{e}}_\mathbf{1} + \mathbf{J_2 P_1 z} \tag{3}$$

By solving this last equation for $\mathbf{z}$, and introducing the computed $\mathbf{z}$ in (2), we finally get:

$$\dot{\mathbf{q}} = \mathbf{J_1^+} \dot{\mathbf{e}}_\mathbf{1} + \mathbf{P_1} (\mathbf{J_2 P_1})^+ (\dot{\mathbf{e}}_\mathbf{2} - \mathbf{J_2 J_1^+} \dot{\mathbf{e}}_\mathbf{1}) \tag{4}$$

Since $\mathbf{P_1}$ is Hermitian and idempotent (it is a projection operator), (4) can be written:

$$\dot{\mathbf{q}} = \mathbf{J_1^+} \dot{\mathbf{e}}_\mathbf{1} + \widetilde{\mathbf{J_2}}^+ \widetilde{\dot{\mathbf{e}}_\mathbf{2}} \tag{5}$$

where $\widetilde{\mathbf{J_2}} = \mathbf{J_2 P_1}$ is the limited Jacobian of the task $\mathbf{e_2}$, giving the available range for the secondary task to be performed without affecting the first task, and $\widetilde{\dot{\mathbf{e}}_\mathbf{2}} = \dot{\mathbf{e}}_\mathbf{2} - \mathbf{J_2 J_1^+} \dot{\mathbf{e}}_\mathbf{1}$ is the secondary task function, without the part $\mathbf{J_2 J_1^+} \dot{\mathbf{e}}_\mathbf{1}$ of the job already accomplished by the first task. A very good intuitive explanation of this equation is given in [1].

B. Extending redundancy formalism for several tasks

Let $(\mathbf{e_1}, \mathbf{J_1})$ ... $(\mathbf{e_n}, \mathbf{J_n})$ be n tasks. We want to extend (5) to these n tasks. Task $\mathbf{e_i}$ should not disturb task $\mathbf{e_j}$ if $i > j$. A recursive extension of (5) is proposed in [37]:

$$\begin{cases} \dot{\mathbf{q}}_\mathbf{0} = 0 \\ \dot{\mathbf{q}}_\mathbf{i} = \dot{\mathbf{q}}_\mathbf{i-1} + (\mathbf{J_i P_{i-1}^A})^+ (\dot{\mathbf{e}}_\mathbf{i} - \mathbf{J_i} \dot{\mathbf{q}}_\mathbf{i-1}), \quad i = 1..n \end{cases} \tag{6}$$

where $\mathbf{P_i^A}$ is the projector onto the null-space of the augmented Jacobian $\mathbf{J_i^A} = (\mathbf{J_1}, \dots \mathbf{J_i})$ and $\widetilde{\mathbf{J}}_\mathbf{i} = \mathbf{J_i P_{i-1}^A}$ is the limited Jacobian of the task i . The robot articular velocity realizing all the tasks in the stack is $\dot{\mathbf{q}} = \dot{\mathbf{q}}_\mathbf{n}$.

Using directly this recursive equation, a projector has to be computed at each step of the computation. A recursive formula for the computation of the projector is proposed in [1]. We recall this equation here:

$$\begin{cases} \mathbf{P_0^A} = \mathbf{I} \\ \mathbf{P_i^A} = \mathbf{P_{i-1}^A} - \widetilde{\mathbf{J}}_\mathbf{i}^{+} \widetilde{\mathbf{J}}_\mathbf{i} \end{cases} \tag{7}$$

where $\mathbf{I}$ is the identity matrix.

Such a hierarchical structure implies some new singularities in the control [8]. Several solutions have been proposed to avoid these new singularities. The first one would be to use the damped-least-square inverse instead of the classical pseudo-inverse [23], [10]. However, the damped factor is difficult to tune, and is often required to be tuned "by hand". A second solution would be to use the Jacobian $\mathbf{J_i}$ instead of the limited Jacobian $\widetilde{\mathbf{J}}_\mathbf{i}$ when computing the pseudo-inverse in (6). The resulting equation is [25], [8]:

$$\dot{\mathbf{q}}_\mathbf{i} = \dot{\mathbf{q}}_{\mathbf{i-1}} + \mathbf{P_{i-1}^A J_i^{+}} \dot{\mathbf{e}}_\mathbf{i}, \quad i = 1..n \tag{8}$$

Comparing to (6), this solution still preserves the hierarchy. However, the control law of the secondary task is not optimal since the projection operator is not taken into account in the pseudo-inverse. In return, this solution is only subject to the singularities of the full Jacobian $\mathbf{J_n^A}$. Finally, a last solution would be to consider the new singularities as new constraints to deal with during the servo [27] for example using the Gradient Projection Method. Since we also use the GPM for constraint application (see Section III-D), it is easy to combine it to avoid the singularities. Finally, such singularities do not appear if the global task Jacobian $\mathbf{J_n^A}$ is full row rank (*i.e.* the number of rows is equals to the rank). This is the case in particular in the experiments presented in this article, thanks to the use of approximately decoupled sensor-based features [40], that is why we have chosen to use (6) in our implementation.

C. Smooth transition

Usually, the control law is obtained from the following equation that constrains the behavior of the task function:

$$\dot{\mathbf{e}} = \mathbf{f_1}(\mathbf{e}) = -\lambda \mathbf{e} \tag{9}$$

Since $\dot{\mathbf{e}} = \mathbf{J}\dot{\mathbf{q}}$, the control law realizing (9) as best as possible is:

$$\dot{\mathbf{q}} = -\lambda \mathbf{J}^{+} \mathbf{e} \tag{10}$$

where λ is used as a parameter to tune the robot speed. The function $\mathbf{f_1}$ in (9) is chosen by the programmer to link $\dot{\mathbf{e}}$ and $\mathbf{e}$. One generally chooses $\mathbf{f_1}(\mathbf{e}) = -\lambda \mathbf{e}$ to set an exponential decoupled decreasing of the error.

The problem of continuity when changing the task $\mathbf{e}$ is due to the lack of constraints on the initial value of $\dot{\mathbf{e}}$. Let $\mathbf{e_A}$ be a global task, used to drive the robot until time $t = 0$. At this time, the control law switches to a second task $\mathbf{e_B}$. Since $\mathbf{e}$ and $\dot{\mathbf{q}}$ are linearly linked, no continuity guarantee can be ensured on $\dot{\mathbf{q}}$, at time t=0.

Soueres et al. proposed a solution to this problem in [38], [39]. They used a second order linear dynamics instead of (10) to take into account two initial conditions $(\mathbf{e}(0), \dot{\mathbf{e}}(0))$:

$$\ddot{\mathbf{e}} + \alpha \, \dot{\mathbf{e}} + \beta \, \mathbf{e} = 0 \tag{11}$$

where the two parameters α and β are used to control both the robot speed and the length of the transient time response. The main drawback is the difficulty in choosing these two parameters to obtain the desired behavior.

In [25], we have proposed to use a non homogeneous first order differential equation to ensure the continuity and to properly decouple the tuning parameters. The differential equation is

$$\dot{\mathbf{e}} = \mathbf{f_2}(\mathbf{e}) = -\lambda \mathbf{e} + \rho(t) \tag{12}$$

where the non homogeneous part $\rho(t)$ is

$$\rho(t) = e^{-\mu t} \left(\dot{\mathbf{e}_\mathbf{A}}(0) + \lambda \mathbf{e_B}(0) \right) \tag{13}$$

where μ is used to set the length of the transient time, and λ to set the decreasing speed of the error. This differential equation is equivalent to a second order one:

$$\ddot{\mathbf{e}} + (\lambda + \mu) \, \dot{\mathbf{e}} + (\lambda \mu) \, \mathbf{e} = 0 \tag{14}$$

Nevertheless, unlike (α, β), this couple of parameters (λ, μ) is properly decoupled. In particular, the end of the transient time is only set by μ. Indeed, the transient period ends when $\mathbf{f_1}$ (see (9)) and $\mathbf{f_2}$ (see (12)) are numerically equivalent, that is to say when $\rho(t)$ is insignificant compared to $\mathbf{e}(t)$, *i.e.*

$$\delta(t) = \frac{\mathbf{f_1}(t) - \mathbf{f_2}(t)}{||\mathbf{f_1}(t)||} = \frac{\rho(0)}{\lambda} \, e^{-\mu t} << 1 \tag{15}$$

The term δ is exponentially decreasing, with a speed set by μ. The task function $\mathbf{e}(t)$ is equivalent to a decreasing exponential function set by λ. It is simply necessary to choose μ bigger than λ to ensure a short transient time response, in comparison with the decreasing time of the task error. The bigger the value μ, the shorter the transient time, but the stronger the acceleration. Experimentally $\mu = 10 \, \lambda$ is chosen.

Let $(\mathbf{e_1}, \dots, \mathbf{e_n})$ be a stack of n tasks. The decreasing speed of each task is chosen separately by using

$$\dot{\mathbf{e}} = \begin{bmatrix} \dot{\mathbf{e_1}} \\ \vdots \\ \dot{\mathbf{e_n}} \end{bmatrix} = - \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{bmatrix} \begin{bmatrix} \mathbf{e_1} \\ \vdots \\ \mathbf{e_n} \end{bmatrix} = -\Lambda \mathbf{e} \tag{16}$$

Equation (6) can be written as $\dot{\mathbf{q}} = \mathbf{A}\dot{\mathbf{e}}$, where the explicit expression of $\mathbf{A}$ is left to the reader. Using (12) and (16), we deduce the complete expression of the control law computed from a stack of tasks

$$\begin{cases} \dot{\mathbf{q}}_\mathbf{i} = \dot{\mathbf{q}}_{\mathbf{i-1}} + (\mathbf{J_i P_{i-1}^A})^{+} (-\lambda_i \mathbf{e_i} - \mathbf{J_i} \dot{\mathbf{q}}_{\mathbf{i-1}}) \\ \dot{\mathbf{q}} = \dot{\mathbf{q}}_\mathbf{n} + e^{-\mu(t-\tau)} \left(\dot{\mathbf{e}}(\tau) + \Lambda \mathbf{e}(\tau) \right) \end{cases} \tag{17}$$

where τ is the time of the last modification of the stack.

D. The Gradient Projection Method

The control law computed above ensures the decreasing of the tasks in the stack, without taking into account the environment of the robot except the interaction between the target and the sensor. To integrate sensor-based control into a complex robotic system, the control law should also make sure that it avoids undesired configurations, such as for example for an eye-in-hand robotic arm joint limits, visual occlusion, obstacles and kinematic singularities. This is done using the Gradient Projection Method [34], [21], [17]. The experiments presented in Section VII demonstrates the generality of this method, applied in this work for joint-limit, visual-occlusion and obstacle avoidance.

In this approach, the robot moves to satisfy the constraints imposed by the environment. The constraints are described by a cost function. The gradient of this cost function can be considered as an artificial force, pushing the robot away from the undesirable configurations. At each iteration, an artificial force $\mathbf{g}(\mathbf{q})$ is induced by the cost function at the current position. Let us consider the problem:

$$\min V(\mathbf{q}), \quad \mathbf{q} \in \mathbb{R}^k \tag{18}$$

where k is the number of robot joints. The classical solution is to move the robot according to the gradient of the cost function, computed in the articular space.

$$\dot{\mathbf{q}} = \kappa \mathbf{g}(\mathbf{q}) = -\kappa \nabla_{\mathbf{q}}^{\top} V \tag{19}$$

where κ is a positive scalar, used as a gain. Therefore, the cost function is generally expressed in the space of the configuration to avoid (e.g. the cost function of visual-occlusion constraint is generally expressed in the image space). Let $\mathbf{\Phi}$ be a parametrization of this space. The cost function is now $V_{\mathbf{\Phi}} = V(\mathbf{\Phi}(\mathbf{q}))$. The corresponding artificial force is given by [29]

$$\mathbf{g}_{\mathbf{\Phi}}(\mathbf{q}) = -\Big(\frac{\partial \mathbf{\Phi}}{\partial \mathbf{q}}\Big)^{+} \nabla_{\mathbf{\Phi}}^{\top} V_{\mathbf{\Phi}} \tag{20}$$

where we can note the use of the Jacobian pseudoinverse. Classical methods propose generally to use simply the transpose of the Jacobian, the artificial force being then $\mathbf{g}_{\mathbf{\Phi}}(\mathbf{q}) = -\Big(\frac{\partial \mathbf{\Phi}}{\partial \mathbf{q}}\Big)^{\top} \nabla_{\mathbf{\Phi}}^{\top} V_{\mathbf{\Phi}}$. Since the pseudoinverse provides the least-square solution, the resulting artificial force (20) is the most efficient one at equivalent norm.

Considering now several minimization problems $V^i = V^i_{\mathbf{\Phi}_i}$, where $\mathbf{\Phi}_i$ are different parametrizations. The global cost function can be written:

$$V = \sum_i \gamma_i V^i_{\mathbf{\Phi}_i} \tag{21}$$

where the scale factors γ_i are used to adjust the relative influence of the different forces. The force realizing a trade-off between these constraints is thus:

$$\mathbf{g} = \sum_i \gamma_i \ \mathbf{g}^i_{\mathbf{\Phi}_i} = \sum_i \gamma_i \ \Big(\frac{\partial \mathbf{\Phi}_i}{\partial \mathbf{q}}\Big)^{+} \nabla_{\mathbf{\Phi}_i}^{\top} V^i_{\mathbf{\Phi}_i} \tag{22}$$

We will see in Section VI the complete definition of the cost functions V for several classical constraints.

E. Final control law

The gradient $\mathbf{g}$ defined in (22) is used as the last task of the stack. It has thus to be projected onto the null space of each task into the stack. Using (17), the complete control law is finally

$$\dot{\mathbf{q}} = \dot{\mathbf{q}}_{\mathbf{n}} + e^{-\mu(t-\tau)} \ \big(\dot{\mathbf{e}}(\tau) + \Lambda \mathbf{e}(\tau)\big) - \kappa \mathbf{P}^{\mathbf{A}}_{\mathbf{n}} \mathbf{g} \tag{23}$$

Therefore, the realization of the constraints depends on two factors. First of all, it depends on the projector $\mathbf{P}^{\mathbf{A}}_{\mathbf{n}}$. When the stack is almost empty, the rank of $\mathbf{P}^{\mathbf{A}}_{\mathbf{n}}$ is high, and the gradient is not much modified. However when the rank decreases near zero (that is when the stack is almost full), the gradient is highly disturbed, especially if the favorite vector direction of the gradient $\mathbf{g}$ does not belong to the range of $\mathbf{P}^{\mathbf{A}}_{\mathbf{n}}$. Of course, when the stack is full, the projector becomes $\mathbf{0}$. The gradient is thus not taken into account any more, and nothing is done to take the constraints into account. The second factor is the gain κ, which defines the influence of the avoidance in the global control law. The choice of this parameter is very important. Indeed, if κ is too small, the gradient force may be too small to respect the constraints. Besides, if κ is too high, some overshoot can occur in the computed velocity. Methods that set this parameter automatically exist (for example [6] for joint-limit avoidance). However it is difficult to generalize to an arbitrary number of additional constraints simultaneously. Moreover, these methods do not provide any solution to the problem due to the rank of $\mathbf{P}^{\mathbf{A}}_{\mathbf{n}}$.

Instead, when the gradient projection method cannot be applied efficiently, we propose to select the subtask of the stack which prevents the control to respect the constraints, and to remove it from the stack. This solution is detailed in the next section.

IV. Using a stack controller

In this section, a controller that removes a subtask from the stack when necessary is proposed. As already explained, a subtask has to be removed from the stack when the current control law is violating one of the constraint to be respected (for example the robot nearly reaches a joint limit). Two criteria have to be built, the first one to decide when a subtask should be removed, the second to choose which subtask to remove.

A. When to remove a subtask ?

The chosen criterion simply consists in determining the effect of the current control law by performing a prediction step before sending the computed velocity to the robot. Let $\mathbf{q}(t)$ be the current articular position of the robot. The predicted position $\widehat{\mathbf{q}}(t+1)$ is given by

$$\widehat{\mathbf{q}}(t+1) = \mathbf{q}(t) + \Delta t \dot{\mathbf{q}} \tag{24}$$

where $\dot{\mathbf{q}}$ is the control law, computed using (23) and Δt can be seen as a gain. A subtask has to be removed from the stack if $V(\widehat{\mathbf{q}}(t+1))$ is above a fixed threshold, where V is the cost function representing the constraints introduced in (21).

B. Which subtask to remove ?

The idea is to detect which subtask induces the most critical conflict with the current projected gradient. We propose two criteria to be computed for each subtask. The subtask to remove is the one corresponding to the maximum (or the minimum, in case of the second criterion) of the values computed. Using both criteria simultaneously gives a more reliable choice. In the following, we present the two criteria, for a subtask $\mathbf{e_i}$, whose Jacobian is $\mathbf{J_i}$, and for an avoidance gradient $\mathbf{g}(\mathbf{q})$.

1) First criterion: The first criterion compares directly the direction of the velocity induced by the subtask, and the one induced by the avoidance gradient. The subtask to remove is the one whose velocity direction corresponds to the opposite of the gradient direction (see Fig. 2(a)). This is done by computing the inner product of the two velocities projected in the same space. The most logical common space seems to be the space of articular velocities. Criterion $\mathcal{C}_\mathbf{1}$ is thus

$$\mathcal{C}_\mathbf{1} = - < \mathbf{J_i^+ e_i} | \mathbf{g} > \tag{25}$$

Another common space can be used, such as the space of the task, using $\mathcal{C}_\mathbf{1b} =< \mathbf{e_i} | \mathbf{J_i g} >$. In this case, the common space depends on each subtask. The experiments have shown that the behavior using any of these criteria is very similar.

This first criterion depends linearly of the task function $\mathbf{e_i}$. If the subtask is nearly completed ($\mathbf{e_i}$ is very low), the criterion is very low. We have experimentally noticed that, using (25), the task controller always removes the last subtask added. We thus use a normalized criterion

$$\mathcal{C_1}' = \frac{1}{||\mathbf{e_i}||}\mathcal{C}_\mathbf{1} \tag{26}$$

Using this last definition, the choice is only based on the velocity direction, and no longer on the velocity norm. Therefore, when the velocity induced by a subtask is very low, the normalization is equivalent to a division by a nearly zero value. That can produce unstable results. The next criterion solve this problem.

2) Second criterion: To compute the final control law, the gradient is projected onto the null space of each subtask. The second criterion computes the contribution of each subtask to this projection. The idea is to remove the subtask whose contribution disrupts the most the constraint (see Fig. 2(b)). The criterion is defined by:

$$\mathcal{C}_\mathbf{2} = ||\mathbf{P_i g}|| \tag{27}$$

where $\mathbf{P_i} = \mathbf{I} - \mathbf{J_i^+ J_i}$ is the projection operator onto the null space of the subtask. Since $\mathbf{P_i}$ is a projection operator, for all vector $\mathbf{x}$, $||\mathbf{P_i x}|| \leq ||\mathbf{x}||$. The less the gradient is in the null space of the subtask, the more it is disturbed, the smaller the value of the criterion. The subtask to be removed is thus the one corresponding to the minimum of $\mathcal{C}_\mathbf{2}$.

3) Another way to compute the second criterion: Another idea is to check if the gradient vector is in the null space of the control law due to the subtask. This subspace is given by (2): it is the range of $\mathbf{J_i^+}$. Consider a basis $(\mathbf{v_1} \dots \mathbf{v_k})$ of the range of $\mathbf{J_i^+}$ (where r is the rank of $\mathbf{J_i^+}$). The criterion is the norm of the gradient, projected in the range of $\mathbf{J_i^+}$

$$\mathcal{C}_\mathbf{2b} = || \sum_{i=1}^{r} (\mathbf{g}^\top \mathbf{v_i}) \mathbf{v_i} || \tag{28}$$

Let us prove that $\mathcal{C}_\mathbf{2}$ and $\mathcal{C}_\mathbf{2b}$ are equivalent. The projection operator does not depend on the basis of its range. Let $\mathbf{V}$ be the basis of SVD of $\mathbf{J_i}$:

$$\mathbf{J_i} = \mathbf{U S V}^\top \tag{29}$$

The singular values are ordered such that $\mathbf{V} = (\mathbf{V_0} \quad \mathbf{V_1})$ where the vectors of $\mathbf{V_0}$ (respectively of $\mathbf{V_1}$) correspond to the null (respectively to the non null) singular values. The third criterion can be thus written as

$$\mathcal{C}_\mathbf{2b} = ||\mathbf{V_1} {\mathbf{V_1}}^\top \mathbf{g}|| \tag{30}$$

Using the SVD, (27) can be written as

$$\mathcal{C}_\mathbf{2} = ||\mathbf{J_i^+ J_i g}|| = ||\mathbf{V_0} {\mathbf{V_0}}^\top \mathbf{g}|| = ||(\mathbf{I} - \mathbf{V_1} {\mathbf{V_1}}^\top)\mathbf{g}|| \tag{31}$$

$\mathcal{C}_\mathbf{2}$ is minimal when $\mathcal{C}_\mathbf{2b}$ is maximal. In fact, $\mathcal{C}_\mathbf{2}$ checks if the gradient is not in the null space of the Jacobian, while $\mathcal{C}_\mathbf{2b}$ checks if the subtask is in the range of the pseudo inverse of the Jacobian, which is equivalent. The experiments confirm that the behaviors using the two criteria are the same. We thus will consider only criterion $\mathcal{C}_\mathbf{1}$ and $\mathcal{C}_\mathbf{2}$ to decide which subtask to remove when it is necessary.

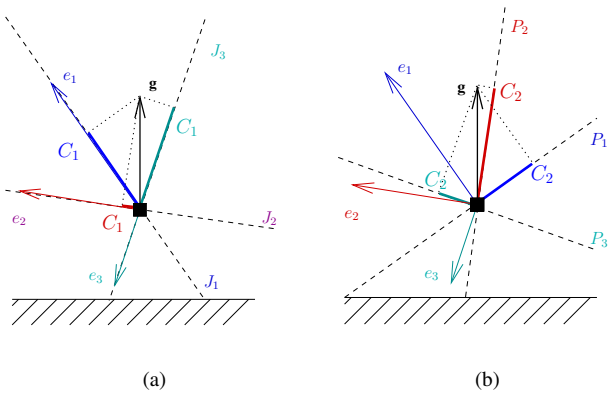


Fig. 2. Computation example for the two criteria. Three subtasks are in the stack. The robot is located at the starting point of the three task vectors. The constraint is represented by an obstacle (the hatched line), close to the robot. The corresponding avoidance gradient is $\mathbf{g}$. Intuitively, the subtask that drives the robot into the obstacle is the green one. **(a)** Criterion $\mathcal{C}_\mathbf{1}$. The maximal criterion is $\mathcal{C}_\mathbf{1}(\mathbf{e_3})$, in green ($\mathcal{C}_\mathbf{1}(\mathbf{e_1})$ is negative and $\mathcal{C}_\mathbf{1}(\mathbf{e_2})$ is nearly zero). **(b)** Criterion $\mathcal{C}_\mathbf{2}$. Projectors $\mathbf{P_1}$, $\mathbf{P_2}$ and $\mathbf{P_3}$ are represented by their vectorial directions (orthogonal to the task vector). The gradient is projected onto these lines. The minimal is $\mathcal{C}_\mathbf{2}(\mathbf{e_3})$ as requested. In the case of this criterion, the sign of the subtask control law is not taken into account. The criterion value for $\mathbf{e_1}$ is very close to the value for $\mathbf{e_3}$ ($\mathbf{e_1}$ nullifies the gradient projection as $\mathbf{e_3}$ does, even if it does not drive directly the robot into the obstacle).

V. Push-back controller and Look-ahead controller

The previous controller ensures that the robot is in the free space and does not violate any constraint. The two remaining

controllers are presented in this section. The first one (*push-back controller*) is used to push the removed subtask back in the stack as soon as possible. When the simple couple *remove-add* is not sufficient to reach the desired position, the last controller (*look-ahead controller*) ensures the convergence by pushing the robot out of any local minimum or dead-lock (see Section V-B).

A. Push-back controller

Each subtask outside of the stack has been removed by the stack controller. The subtask can thus be associated to a constraint that has caused the removal. The controller should put the subtask back in the stack as soon as it does not risk to violate the constraint anymore. This is done by a prediction phase. The controller predicts the evolution of the constraint cost-function value with respect to the motion of the robot driven only by the subtask. Let $\mathbf{e_i}$ be a subtask that is not in the stack, $\mathbf{q}_t$ the current articular position, and $\mathbf{\Phi}$ the parameters of the space where the constraint that has caused the removal is defined. The predicted displacement to complete the subtask is

$$\mathbf{\Delta q} = -\mathbf{J_i^+ e_i} \tag{32}$$

The controller predicts that it is safe to put the subtask back in the stack if the intersection between the segment $\mathbf{\Phi}([\mathbf{q}_t, \mathbf{q}_t + \mathbf{\Delta q}])$ and the region where the constraint is violated is empty. This can be mathematically written as

$$\max_{\mathbf{q}\in[\mathbf{q}_t,\mathbf{q}_t+\mathbf{\Delta q}]} \left\{V_{\mathbf{\Phi}(\mathbf{q})}\right\} < V_{max} \tag{33}$$

B. Look-ahead controller

1) When to start: This last controller ensures the convergence of the global algorithm by pushing the robot out of any local minimum or dead-lock. These two situations may occur due to the approximations involved in Controller 2 and 3 that only consider linear approximations of the evolution equations. These linear approximations are equivalent to consider only the local part of the environment closest to the robot. The robot is thus unable to any look-ahead computations and can come to a dead end. This last controller is introduced to give to the robot some look-ahead capabilities. An overview of the controller principle is given by a simple 2D example on Fig. 3. The robot reaches a local minimum when going toward the desired position. The look-ahead controller is activated to leave the attractive area of the local minimum. When the robot leaves the local minimum, the sensor-based control is activated again. If another local minimum is reached, the controller is activated once more, then inactivated again when the new local minimum is left, etc.

2) What to do: When a dead lock or a local minimum is reached, the controller has to introduce a specific task in the stack that is able to move the robot out of the dead end. This kind of problem has already been widely considered in robotic to enlarge the convergence area of local path planning methods [17], [2], [20]. Several solutions can be proposed. A first solution is to compute some open-loop displacement to leave the dead end (for example by introducing some random term in the robot displacement [2]). A second more-reliable solution is to use some additional knowledge about the environment to compute a path that leaves the local minimum. The task to be introduced in the stack by the look-ahead controller is then a sensor-based path following task such as those used in [35], [**?**]. Even if this solution requires a lot of knowledge about the environment, this method is different to the classical path-planning/execution method since 1) path planning is not used in the normal algorithm running, 2) no plan is computed but when a dead end is reached and 3) when needed, path planning is only used in a very short time period only to leave the local minimum, and not to reach the desired position. A last solution is to compute a secondary goal that should be reached before joining the desired position. The task added in the stack is then a sensor-based servo control to this secondary goal.

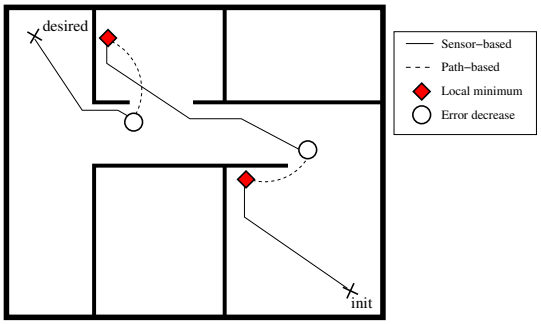


Fig. 3. Reaching the convergence domain. The look-ahead controller starts and stops several times, until the convergence domain is properly reached. The top controller is stopped as soon as the robot reaches a new region, but is reactivated if necessary.

These three solutions are all available and the choice has to be made depending on the application. Since the task to be added by the controller is highly dependent on the application context, it is very uneasy and hardly interesting to generalize it in a mathematical way. In the experiment presented at the end of the article, we have used the last solution (see Section VII-C). A secondary goal is defined in the articular space to escape a local minimum due to non-convex articular structure of the robot.

3) When to stop: Finally, the look-ahead controller has also to decide when to remove the specific task from the stack, and let the normal execution start again. The specific task should be stopped as soon as the robot reaches the convergence domain of the sensor-based main task. It is very difficult to determine if the robot is into the convergence domain since generally no analytical description of the domain can be written. We rather compute if the robot has left the convex sub-area where the removed sensor-based subtask was unable to converge. This can be obtained by considering the progress of the sensor-based subtask. In the example depicted in Fig. 4, the task error increases when going round the obstacle, since the robot is leaving a local minimum. When the local minimum is left, the sensor-based task error starts decreasing. The look-ahead controller is thus inactivated when the subtask error is decreasing, that is to say when $\dot{\mathbf{e}}_\mathbf{i}$ is negative. To prevent any false detection due to measure noise, the error derivative is integrated onto several iteration. The controller inactivation

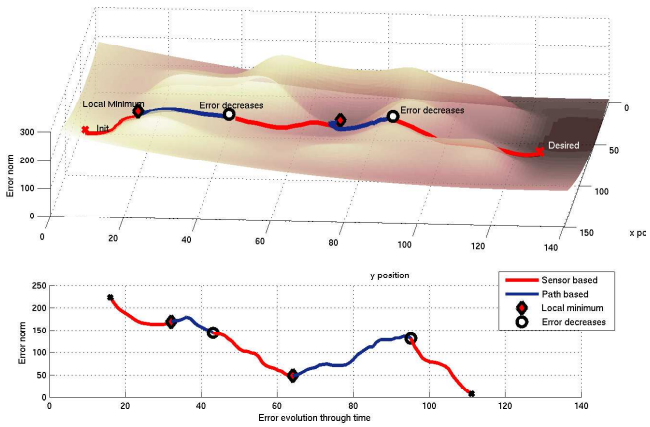


Fig. 4. Example of the interest of the look-ahead controller for the execution environment presented in Fig. 3 (a) The trajectory of the robot in the plane along with the value of the error for each position (b) Evolution of the error versus time for the same robot execution. The look-ahead controller is activated a first time when a local minimum is reached. The error increases when leaving the local minimum. The controller is stopped as soon as it is detected that the sensor error is decreasing. The similar sequence is applied when a second local minimum is reached .

criterion is thus :

$$\mathcal{C} = \int_{t-\Delta t}^{t} \dot{\mathbf{e}}(\mathbf{q}_t)dt < 0 \tag{34}$$

where Δt is a parameter that tunes the length of the integration time interval (this parameter is not very important since it is just used to prevent the false detection due to velocity peak. Δt is typically set to five iterations in the experiments). Finally after integration of the derivative, the inactivation criterion can be written

$$\mathcal{C} = \mathbf{e}(t) - \mathbf{e}(t - \Delta t) < 0 \tag{35}$$

VI. Implementation in Visual servoing

The only hypothesis done to realize the work presented in the previous sections is that the main task is a task function [35]. The proposed control scheme is thus very general and can be applied in several domains for closed-loop control. We have implemented our approach using the visual-servoing framework [11], [15] to control a six-DOF eye-in-hand robot. The environment constraints we have considered to validate the proposed architecture are articular joint-limits, occlusion and obstacles in the Cartesian space. In this section, the visual servoing framework is first quickly recalled. The visual features chosen for the servo are image moments [40]. We then present the cost functions used to represent the constraints we have considered.

A. Four subtasks to constrain the six DOF

The subtask functions $\mathbf{e_i}$ used in the remainder of the text are computed from visual features [11]:

$$\mathbf{e_i} = \mathbf{s_i} - \mathbf{s_i^*} \tag{36}$$

where $\mathbf{s_i}$ is the current value of the visual features for subtask $\mathbf{e_i}$ and $\mathbf{s_i^*}$ their desired value. The interaction matrix $\mathbf{L_{s_i}}$ related to $\mathbf{s_i}$ is defined so that $\dot{\mathbf{s_i}} = \mathbf{L_{s_i}}\mathbf{v}$, where $\mathbf{v}$ is the instantaneous camera velocity. From (36), it is clear that the interaction matrix $\mathbf{L_{s_i}}$ and the task Jacobian $\mathbf{J_i}$ are linked by the relation:

$$\mathbf{J_i} = \mathbf{L_{s_i}}\mathbf{M}\mathbf{J_q} \tag{37}$$

where the matrix $\mathbf{J_q}$ denotes the robot Jacobian ($\dot{\mathbf{r}} = \mathbf{J_q}\dot{\mathbf{q}}$) and $\mathbf{M}$ is the matrix that relates the variation of the camera velocity $\mathbf{v}$ to the variation of the chosen camera pose parametrization ($\mathbf{v} = \mathbf{M}\dot{\mathbf{r}}$).

In order to obtain a better and easier control over the robot trajectory, approximatively decoupled subtasks are chosen. As explained in the previous parts, there is no need to choose them perfectly independent, thanks to the redundancy formalism. The visual features are derived from the image discrete moments. The discrete moments are computed from a set of relevant points of the image target. At each iteration, let $\mathbf{x_i} = (x_i, y_i)$ be the position of the points in the image. The moment $m_{i,j}$ of the object is defined by

$$m_{i,j} = \sum_{k=1}^{N} x_k^i y_k^j \tag{38}$$

The first subtask $\mathbf{e_g}$ is based on the position of the center of gravity. It is defined by:

$$(x_g, y_g) = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}) \tag{39}$$

The second subtask $\mathbf{e_Z}$ uses the area a of the object in the image to control the range between the robot and the target [40]:

$$a_n = \sqrt{\frac{a^*}{a}} \tag{40}$$

where a^* is the value of a computed from the desired image. To decouple the other subtasks, the centered moments are used. The centered moment $\mu_{i,j}$ of a set of points is

$$\mu_{i,j} = \sum_{k=1}^{N} (x_k - x_g)^i \cdot (y_k - y_g)^j \tag{41}$$

The third subtask $\mathbf{e}_\alpha$ is used to correctly angle the object in the image. It uses the orientation of the object in the image, defined by [40]:

$$\alpha = \frac{1}{2} Arctan(\frac{2\mu_{11}}{\mu_{20} - \mu_{02}}) \tag{42}$$

The last subtask $\mathbf{e_R}$ uses third order moments to decouple v_x from ω_y and v_y from ω_x. The moments choice is less intuitive than for the three fist tasks. The reader is invited to refer to [40] for more details.

B. Avoidance control laws

The avoidance laws are computed using (22). We propose here an implementation for joint-limit, occlusion and obstacle avoidance. For each constraints, we give the cost function. When necessary, the Jacobian matrix used to pass from the space where the constraint is defined to the articular space is also provided.

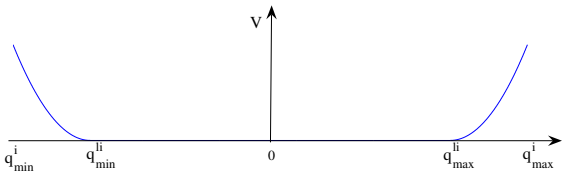


Fig. 5. Cost function of the joint limits avoidance for an articulation

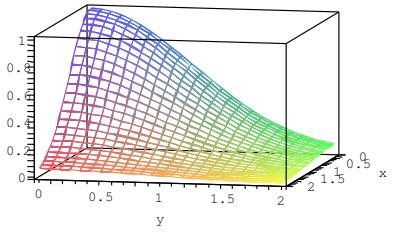


Fig. 6. Cost function of the visual-occlusion avoidance in the image space

1) Joint-limit avoidance: The cost function for joint-limit avoidance is defined directly in the articular space. It reaches its maximal value near the joint limits, and it is nearly constant (so that the gradient is nearly zero) far from the limits.

The robot lower and upper joint limits for each axis i are denoted $\bar{q}_i^{\min}$ and $\bar{q}_i^{\max}$. The robot configuration $\mathbf{q}$ is said acceptable if, for all i, $\mathbf{q}_i \in [\bar{q}_{\ell i}^{\min}, \bar{q}_{\ell i}^{\max}]$, where $\bar{q}_{\ell i}^{\min} = \bar{q}_i^{\min} + \rho\bar{q}_i$, $\bar{q}_{\ell i}^{\max} = \bar{q}_i^{\max} - \rho\bar{q}_i$, $\bar{q}_i = \bar{q}_i^{\max} - \bar{q}_i^{\min}$ is the length of the domain of the articulation i, and ρ is a tuning parameter, in $[0, 1/2]$ (typically, $\rho = 0.1$). $\bar{q}_{\ell i}^{\min}$ and $\bar{q}_{\ell i}^{\max}$ are activation thresholds. In the acceptable interval, the avoidance force should be zero. The cost function V^{jl} is thus given by (see Fig. 5) [6]:

$$V^{jl}(\mathbf{q}) = \frac{1}{2}\sum_{i=1}^{n} \frac{{\delta_i}^2}{\Delta\bar{q}_i} \tag{43}$$

where

$$\delta_i = \begin{cases} \mathbf{q}_i - \bar{q}_{\ell i}^{\min}, & \text{if } \mathbf{q}_i < \bar{q}_{\ell i}^{\min} \\ \mathbf{q}_i - \bar{q}_{\ell i}^{\max}, & \text{if } \mathbf{q}_i > \bar{q}_{\ell i}^{\max} \\ 0, & \text{else} \end{cases}$$

2) Occlusion avoidance: Occlusion avoidance depends on data extracted from the image. An image processing step detects the occluding object (if any). The avoidance law should maximize the distance d between the occluding object and the visual target that is used for the main task. Let d_x and d_y be the x and y coordinates of the distance between the target and the occluding object ($d = \sqrt{d_x^2 + d_y^2}$) and $\mathbf{x_a}$ be the point of the occluding object that is the closest to the target.

The cost function V^{occ} is defined in the image space, so that it is maximal when d is 0, and nearly 0 when d is high (see Fig. 6). Like in [28], we simply choose:

$$V^{occ}(d) = e^{-\beta d^2} \tag{44}$$

The parameter β is arbitrary and can be used to tune the effect of the avoidance control law. The gradient in the image space is obtained by a simple calculation:

$$\nabla_{\mathbf{x}}^{\top} V^{occ} = \begin{pmatrix} -2\beta d_x e^{-\beta d^2} \\ -2\beta d_y e^{-\beta d^2} \end{pmatrix} \tag{45}$$

The artificial force that avoids the occlusions can be now computed using (20). The transformation from the image space to the articular space is given by [29]:

$$\mathbf{g}^{occ} = -\Big(\frac{\partial \mathbf{x}}{\partial \mathbf{r}}\frac{\partial \mathbf{r}}{\partial \mathbf{q}}\Big)^{+} \nabla_{\mathbf{x}}^{\top} V^{occ} = -(\mathbf{L_x M J_q})^{+} \nabla_{\mathbf{x}}^{\top} V^{occ} \tag{46}$$

where $\mathbf{M}$ and $\mathbf{J_q}$ are the transformation matrices defined in (37), and $\mathbf{L_x}$ is the well-known interaction matrix related to the image point $\mathbf{x_a}$.

3) Obstacle avoidance: The obstacles are defined in the Cartesian 3D space. We propose to use the rotational potential first proposed in [16] extended from the case of a 2D non-holonomic robot to the 3D Cartesian space.

Let P_0 be the nearest point of the obstacle to the robot. Let $\mathbf{n_0}$ be the normal to the obstacle at P_0. To apply the formalism defined in [16], the 3D Cartesian space should be restricted to a plane. Let $\mathbf{v}$ be the current translational velocities components of the camera. We consider only the plane $(P_0, \mathbf{n_0}, \mathbf{v})$. Let $\mathbf{t_0}$ be the only tangent to the obstacle at P_0 so that the plane $(P_0, \mathbf{n_0}, \mathbf{v})$ and $(P_0, \mathbf{n_0}, \mathbf{t_0})$ are equal. Let $\mathcal{F}_\mathbf{0}$ be the orthonormal frame $(P_0, \mathbf{n_0}, \mathbf{t_0}, \mathbf{z_0})$, where $\mathbf{z_0}$ is the unique vector so that $\mathcal{F}_\mathbf{0}$ is orthonormal. Figure 7 sums up all these vector definitions.

The coordinates of a point in frame $\mathcal{F}_\mathbf{0}$ are noted $\mathbf{r_0} = (n, t, z)$. The potential function in $\mathcal{F}_\mathbf{0}$ is defined by:

$$V_{\mathbf{r_0}}^{obs} = \begin{cases} \frac{1}{2}k_1\big(\frac{1}{n} - \frac{1}{\bar{n}}\big)^2 + \frac{1}{2}k_2 t(n-\bar{n})^2 & \text{if } n < \bar{n} \\ 0 & \text{otherwise} \end{cases} \tag{47}$$

where k_1 and k_2 are tuning parameters (typically, $k_1 >> k_2$), and $\bar{n}$ is the maximal distance above which the obstacle is not taken into account. The function is shown on Fig. 8

The gradient is obtained directly from (47). The corresponding Jacobian is

$$\frac{\partial \mathbf{r_0}}{\partial \mathbf{q}} = \frac{\partial \mathbf{r_0}}{\partial \mathbf{r}}\frac{\partial \mathbf{r}}{\partial \mathbf{q}} = {}^0\mathbf{R}_c\big(\mathbf{I_3} \;\; \mathbf{0_3}\big)\mathbf{J_q} \tag{48}$$

where ${}^0\mathbf{R}_c$ is the rotation from frame $\mathcal{F}_\mathbf{0}$ to the camera frame, $\mathbf{I_3}$ and $\mathbf{0_3}$ are the identity and the null matrix in dimension three and $\mathbf{J_q}$ is the articular Jacobian.

VII. Experiments and Results

We present in this section the experiments realized to validate the proposed method. The experiments have been realized using a six DOF eye-in-hand Gantry robot. The robot has to position with respect to a visual target. Since the main purpose of these experiments was the robot control, the image processing part has been simplified by using a very easy target composed of four white dots (see Fig. 9). All the computations have been done on a classical 2.0GHz PC, with a standard IEEE1394 firewire camera. The control loop is at video rate (that is 25Hz), even if no special effort has been done in the implementation to optimize this point.

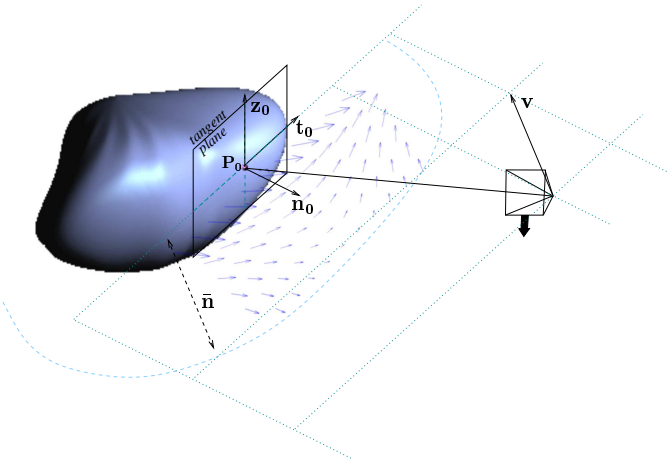


Fig. 7. Frame $\mathcal{F}_\mathbf{0}$. The origin point of $\mathcal{F}_\mathbf{0}$ is the nearest point of the obstacle to the camera, noted $\mathbf{P_0}$. Vector $\mathbf{n_0}$ is the normal to the obstacle at $\mathbf{P_0}$. Among all the tangent vectors to the obstacle at $\mathbf{P_0}$ we choose $\mathbf{t_0}$ so that $\mathbf{n_0} \times \mathbf{t_0}$ and $\mathbf{n_0} \times \mathbf{v}$ are equal. The last vector $\mathbf{z_0}$ of $\mathcal{F}_\mathbf{0}$ is chosen so that the frame is orthonormal. In the plane $\mathbf{P_0}, \mathbf{n_0}, \mathbf{t_0}$, the cost function is defined by (47). Its gradient vector field is drawn on the figure.

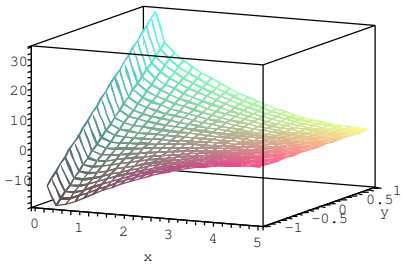
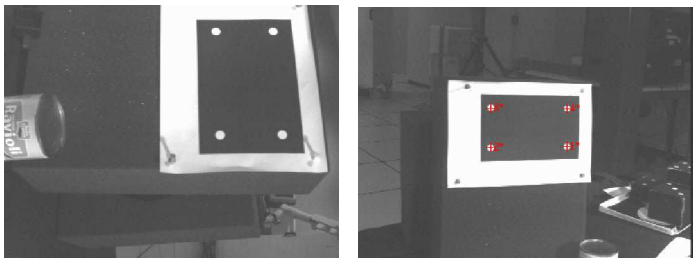


Fig. 8. Cost function of the obstacle avoidance in plane $(P_0, \mathbf{n_0}, \mathbf{t_0})$



(a) (b)

Fig. 9. Snapshot using the eye-in-hand embedded camera. These pictures have been taken during the second experiment. (a) Initial image (b) Desired image

Three set of experiments are presented in the following, varying the constraints taken into account in the control law. Since the positioning task uses all the robot DOF, no redundancy is available for the additional constraints with the classical formalism. The robot is thus unable to reach the goal using classical control laws because of the constraints, but always manages to complete the task using the proposed method. In the first experiment, the robot has to avoid occlusions due to a moving object passing between the camera and the target, and to deal with its joint limits at the same time. In the second experiment, some obstacles have been put into the work space of the robot. The robot has thus to avoid simultaneously the obstacles themselves and the occlusions they can cause to complete the positioning. Since the detection of such obstacles by image processing is a complex problem, this last experiment has been realized in simulation only. The last experiment takes only the joint limits into account. The robot starts in a non-convex part of its joint-limit space, so that the look-ahead controller is required to complete the positioning. This is a typical example of the interest of the look-ahead controller.

A. First experiment

In this experiment, the robot starts very close to the desired position. It is asked to maintain this position. During the servo, an object moves between the target and the camera, inducing a visual occlusion. The robot has to reactively avoid this occlusion, and also its joint limits, since the first avoidance motion drives the robot in it. Finally, when the moving object has passed on, the robot has to reach the desired position, as required by the main positioning task.

The experiment is summed up from Fig. 10 to 15. Each action on the stack (add or remove) is represented by a vertical straight line on each graph. The events are referenced from (1) to (5) on Fig. 10.

At Event (1), the controller predicts a visual occlusion, and removes thus the optimal subtask to take the occlusion constraint into account (Task $\mathbf{e_R}$, see Fig. 11). The robot then escapes the visual occlusion by mainly rotating around the target. As shown on Fig. 12, this motion drives the robot into its joint limits. Once again, the controller predicts the collision, and removes successively Tasks $\mathbf{e}_\alpha$ and $\mathbf{e_Z}$ to deal with the joint-limits avoidance. At Event (3), the occluding object stops its motion but does not move away. An equilibrium is reached. Controller 3 thus decides to put the removed subtasks back in the stack (the tasks are put back in the removal order, last out, first in). Since the occluding object has not moved away, the subtasks have to be removed once more, until the occluding object moves away (Event (4)). The subtasks are then put back, and the robot moves to reach the desired position. During the motion, it nearly reaches one of its joint limits (see Fig. 13). A subtask is thus temporarily removed from the subtask during few iterations (Event (5)): according to the criterion values, the optimal task $\mathbf{e_z}$ is removed (see Fig. 11).

B. Second experiment

For this experiment, an obstacle is present into the robot workspace. The robot has to reach a desired position, avoiding

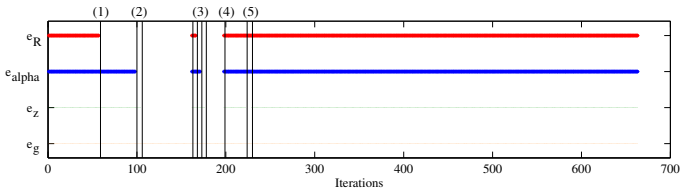


Fig. 10. Experiment A: Event and activation graph
Each action on the stack (add or remove) is represented by a vertical straight line on each graph. The relevant actions are regrouped and numbered to be referenced in the text.
At start all the tasks are in the stack. The task order is $[\mathbf{e_G}, \mathbf{e}_\alpha, \mathbf{e_Z}, \mathbf{e_R}]$. Controller 2 predicts an occlusion at Event (1) and Task $\mathbf{e_R}$ is removed. The controller then predicts a collision with the joint limits at Event (2), and removes successively Tasks $\mathbf{e}_\alpha$ and $\mathbf{e_Z}$. Controller 3 puts the three subtasks back in the stack at Event (3). The stack order is then $[\mathbf{e_G}, \mathbf{e_Z}, \mathbf{e}_\alpha, \mathbf{e_R}]$. However since the occluding object has not moved away yet, Controller 2 removes the subtasks from the stack again (one task at each iteration during three iterations). The occluding object moves away at Event (4). All the subtasks are then put back (same stack order), and the robot moves to join its desired position. During the motion, it nearly reaches its joint limits at Event (5), which causes Controller 2 to remove temporarily Task $\mathbf{e_Z}$. After Event (5), the stack order is $[\mathbf{e_G}, \mathbf{e}_\alpha, \mathbf{e_R}, \mathbf{e_Z}]$.

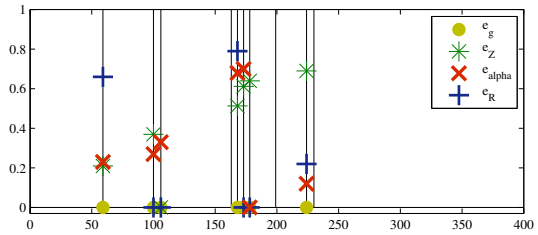


Fig. 11. Experiment A: Tasks criteria for removal
The task corresponding to the maximum of the four criteria is removed by Controller 2. The criteria are computed only when Controller 2 removes a task (seven times during this execution). Each time the controller removes a subtask, a clear maximum appears: the selected criteria are properly discriminatory. In this experiment, the criterion for $\mathbf{e_g}$ is forced to zero to always keep the centering task active, since loosing the centering quickly leads to the object visibility loss.

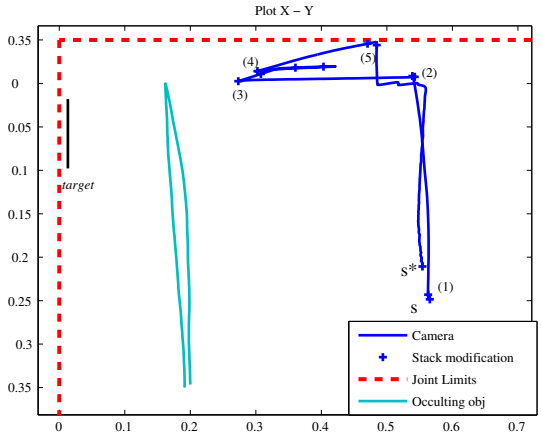


Fig. 12. Experiment A: Camera motion in the Cartesian space (plane X-Y)
The robot nearly reaches the joint limits at point (0.54, 0.04). The occlusion is then avoided by going forward, closer from the visual target.

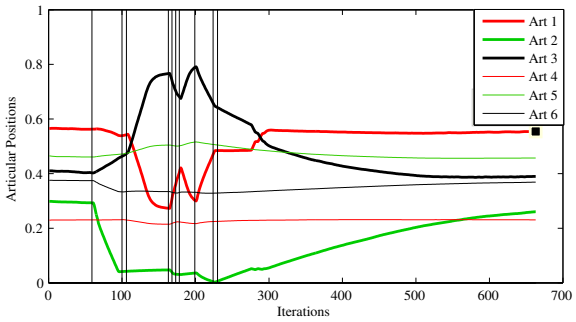


Fig. 13. Experiment A: Articular trajectories
The robot comes very close to its third joint limit at Event (2). It then stays close to the limit since the DOF not used for positioning are used for occlusion avoidance. During the motion back to the desired position (after Event (4)) the third joint limit is nearly reached (Event (5)). It is avoided by removing temporarily Task $\mathbf{e_Z}$ (as can be seen on Fig. 10).

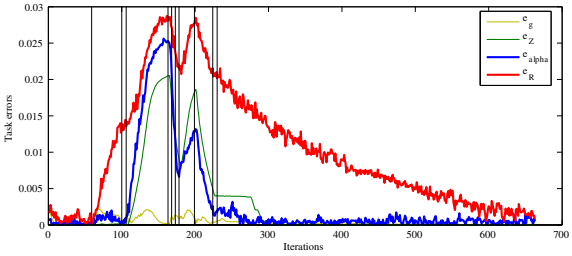


Fig. 14. Experiment A: Tasks error
Task $\mathbf{e_R}$ is removed at Event (1). Tasks $\mathbf{e}_\alpha$ and $\mathbf{e_z}$ are removed at Event (2). Their errors increase from these instants since their corresponding DOF are used for avoidance. They are definitively put back at Event (4) and then decrease until 0.

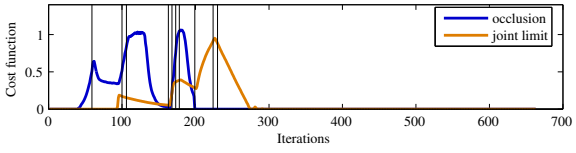


Fig. 15. Experiment A: Occlusion and joint-limit cost functions
The occlusion function increases until Event (1). As soon as a task is removed, the occlusion cost function decreases while the robot is far from the joint limits. At Event (2), the joint-limit function increases. The DOF used for occlusion avoidance is not available any more. The occlusion function increases again until other subtasks are removed, and then decreases. Between Event (3) and (4), the occlusion disappears. During the motion to the desired position, the robot comes closer from its joint limit. The joint-limit function increases until it is avoided (Event (5)).

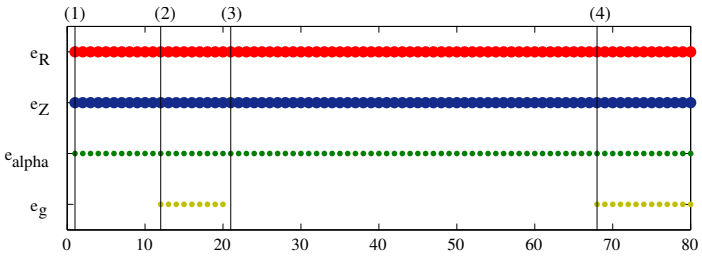
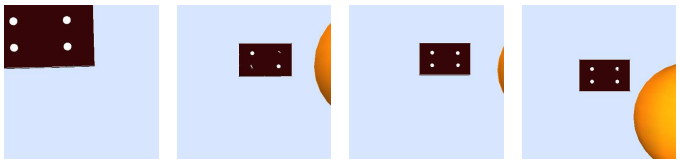


Fig. 16. Experiment B: Event and activation graph
At the beginning the stack order is $[\mathbf{e_G}, \mathbf{e_\alpha}, \mathbf{e_R}, \mathbf{e_Z}]$. Task $\mathbf{e_g}$ is removed at Event (1) to avoid the obstacle according to the criterion values (see Fig. 19). The obstacle is avoided at Event (2) and Task $\mathbf{e_g}$ put back at the top of the stack. An occlusion is predicted at Event (3), and Task $\mathbf{e_g}$ is once more removed (see Fig. 19). It is lately put back at the top of the stack at Event (4) to complete the servo.



(a) (b) (c) (d)

Fig. 17. Experiment B: Snapshots of the eye-in-hand camera
(a) At the initial position (b) When occlusion is predicted (Event (3) (c) When occlusion is avoided (Event (4) (d) At final position. At this position, the obstacle is in the field of view. The prediction is accurate enough to detect that it is harmless.

the obstacle, and also the visual occlusion it can produce. As said previously, this experiment has been realized in simulation since the obstacle detection is a difficult part, which is not the subject of this article.

The events and the corresponding activation of the subtasks are given in Fig. 16. Figures 17 to 20 show respectively some screenshots taken from the simulator during the servo, the Cartesian trajectory of the robot, the criteria evolution and the error of the subtasks.

The camera has mainly to move backward to reach the desired position. However this motion drives the robot into the obstacle. One DOF is freed up by removing Task $\mathbf{e_G}$ from the stack (Event(1)). The camera goes around the obstacle and Task $\mathbf{e_G}$ is put back in the stack. The camera then passes behind the obstacle, which causes an occlusion (Event (3)). Once more, the centering is chosen and removed from the stack. The occlusion can thus be avoided and the robot reaches the position after Task $\mathbf{e_G}$ has been put back at the top of the stack (Event (4)).

C. Third experiment

The last experiment presents the interest of the look-ahead controller. The only constraint considered here is the one imposed by the joint limits. The required motion is mainly a

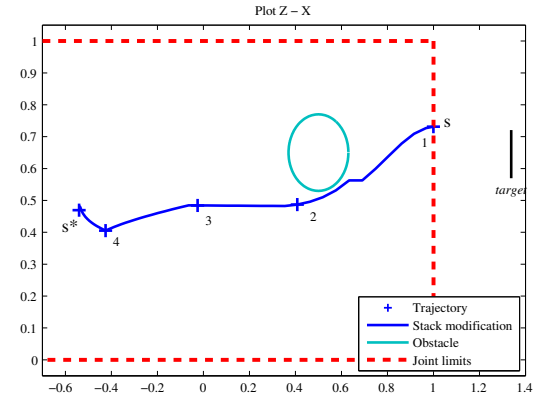


Fig. 18. Experiment B: Camera motion in the Cartesian space (plane X-Z).

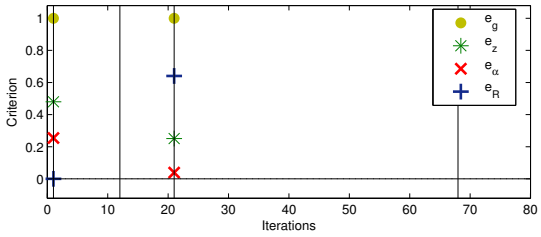


Fig. 19. Experiment B: Tasks criteria for removal

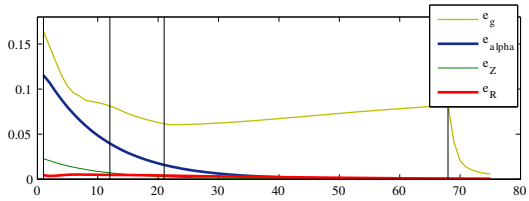


Fig. 20. Experiment B: Tasks error
Only Task $\mathbf{e_g}$ is relaxed during the servo. The other convergences are exponential. Between Event (1) and (2), Task $\mathbf{e_g}$ decreases while it is not in the stack. The avoidance motion corresponds to a centering. However, the decreasing is not exponential (it is faster). It is then not possible to let the subtask in the stack and to avoid the obstacle simultaneously tile Event (5).

Z-rotation of the camera (approximately 60 dg). In that case, the joint limit of the wrist is reached when doing this rotation. The robot has then the opposite 300 dg rotation to realize.

When considering the joint limits, the only local minimum which may occur comes from the non-convex structure of the map between articular and Cartesian spaces. When the robot is stuck near a joint limit in a local minimum, the look-ahead task consists in reaching an intermediate goal, which is simply defined as the opposite joint limits. This task is thus applied without any visual-feedback control (but using the actuator feedback to close the loop). The task function is simply written:

$$\mathbf{e_{art}} = \left(q_i - \bar{q}_i\right) \tag{49}$$

where i is the joint which should be overpassed and $\bar{q}_i$ is the joint upper value $\bar{q}_i^{\max}$ if the robot is stuck near the lower joint limit and $\bar{q}_i^{\min}$ otherwise.

Fig. 21 to 24 sum up the experiment. The wrist joint (Joint 4 in Fig. 22) starts close to the upper joint limit. At the beginning the robot simply realizes the minimization of all tasks (all

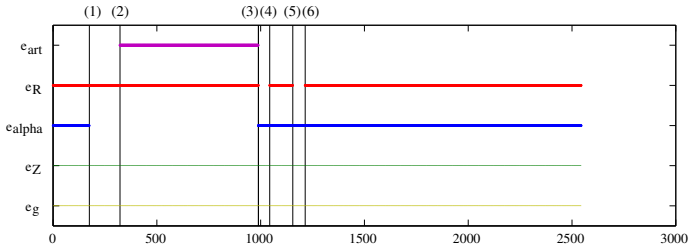


Fig. 21. Experiment C: Event and activation graph
The initial stack order is $[\mathbf{e_G}, \mathbf{e}_\alpha, \mathbf{e_R}, \mathbf{e_Z}]$. Task $\mathbf{e}_\alpha$ is removed at Event (1). The system then converges into a local minimum and the look-ahead controller is activated at Event (2) by adding the joint-value-based task at the top of the stack. This special subtask is then removed from the stack while the subtask $\mathbf{e}_\alpha$ is put back at Event (3). The stack order is then $[\mathbf{e_G}, \mathbf{e_R}, \mathbf{e_Z}, \mathbf{e}_\alpha]$. Almost simultaneously, Task $\mathbf{e_R}$ is removed temporarily a first time and put back at Event (4), and a second time (Event (5) and (6)). The final task order is $[\mathbf{e_G}, \mathbf{e_Z}, \mathbf{e}_\alpha, \mathbf{e_R}]$.

task errors decrease, see Fig. 23). The corresponding motion on Joint 4 is a wrong way rotation: the robot realizes the shortest motion on the wrist torus, disregarding the joint limit. The fourth articular joint value increases, coming closer to the joint limit and the joint limit is nearly reached at Event (1) (see Fig. 22). Controller 2 removes the rotation subtask (see Fig. 24). The robot manages to complete all the other subtasks (Event (2)). The system is then in a local minimum. The look-ahead controller is activated. The fourth joint moves without considering the visual features, and Task $\mathbf{e}_\alpha$ error increases. The error $\mathbf{e}_\alpha$ starts decreasing at Event (3), which corresponds to the detection of the limit of the local minimum attraction domain (34). As soon as the robot leaves this domain, the look-ahead controller is set off, and Task $\mathbf{e}_\alpha$ is put back at the top of the stack. During the final motion, the robot nearly reaches another joint limit. This implies to temporarily remove Task $\mathbf{e_R}$ (Event (4)). Finally, this subtask is quickly put back in the stack (Event (5)).

VIII. Conclusion

In this paper, a general method has been proposed to take into account the constraints due to a real robotic environment such as joint limits, occlusion or obstacles while moving the robot according to a main task with higher priority. The full constraining global task is divided into several subtasks, which can be temporarily removed from the execution in order to free up some DOF for considering the constraints. A complete system has been built that ensures that enough DOF are always available to take the constraints into account, and that the robot completes the full task when it is possible. This system is thus able to provide a convergence in a large blocked-up environment, as path planning does, however reactively and without any global knowledge about the environment. Several set of experiments have shown that this approach is able to converge despite various kinds of constraints until the desired position. Future works will be devoted to the application of such a method to underactuated robots, such as non-holonomic robots, or to highly redundant systems such as mobile manipulators and humanoids.

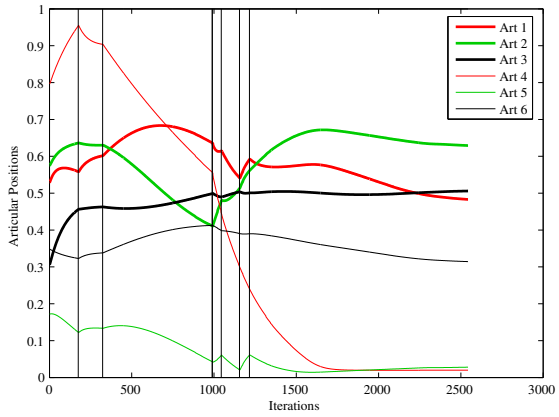


Fig. 22. Experiment C: Articular trajectories
Joint 4 is a wrist with a joint limit between angles 0 dg and 360 dg (corresponding respectively to normalized values 0 and 1). At the beginning, taking only into account the main vision-based task and disregarding the joint limits, the robot try to move *through* this limit: the fourth joint value increases. Since it is not possible to pass through the joint limit, the look-ahead controller is activated from Event (2) to Event (3) to go around the limit. The specific task decreases the joint angular value. The look-ahead controller is switched off at Event (3) since the visual-servoing convergence domain has been reached. The robot then reaches the correct position using a classical vision-based minimization. The fifth joint limit is nearly reached at Events (4) and (5). The accuracy of the controllers finally enable a positioning very close to the joint limits.

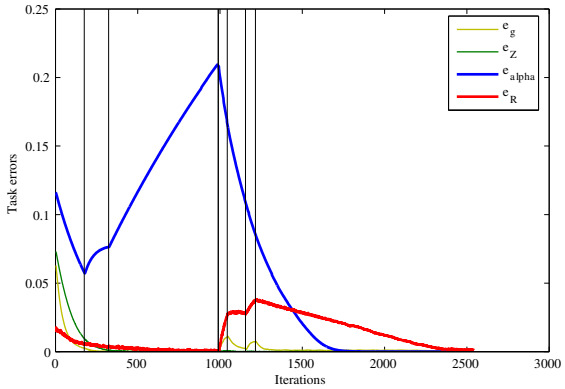


Fig. 23. Experiment C: Tasks error
Task $\mathbf{e}_\alpha$ decreases at the beginning until the robot reaches its joint limit (Event (1)). The subtask error increases then when the robot moves according to the specific task introduced by the top controller. When the robot leaves the local minimum attraction domain (Event (3)), the subtask is put back in the stack and decreases until 0. Task $\mathbf{e_R}$ is also temporarily relaxed to avoid another limit at Event (4) and (5).

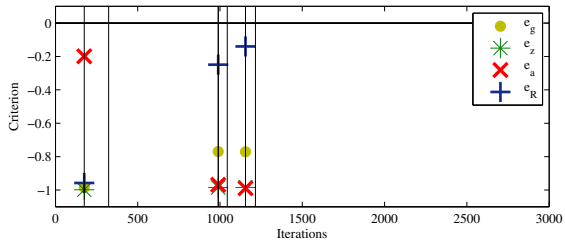


Fig. 24. Experiment C: Tasks criteria for removal

REFERENCES

[1] P. Baerlocher and R. Boulic. An inverse kinematic architecture enforcing an arbitrary number of strict priority levels. *The Visual Computer*, 6(20):402–417, August 2004.

[2] J. Barraquand, B. Langlois, and J.-C. Latombe. Numerical potential field techniques for robot path planning. *IEEE Transactions on Systems, Man and Cybernetics*, 22(2):224–241, March 1992.

[3] O. Brock and O. Khatib. Elastic strips: A framework for motion generation in human environments. *Int. Journal of Robotics Research*, 21(12):1031–1052, December 2002.

[4] T. Chang and R. Dubey. A weighted least-norm solution based scheme for avoiding joints limits for redundant manipulators. *IEEE Trans. on Robotics and Automation*, 11(2):286–292, April 1995.

[5] F. Chaumette. Potential problems of stability and convergence in image-based and position-based visual servoing. In D. Kriegman, G . Hager, and A. Morse, editors, *The Confluence of Vision and Control*, pages 66–78. LNCIS Series, No 237, Springer-Verlag, 1998.

[6] F. Chaumette and E. Marchand. A redundancy-based iterative scheme for avoiding joint limits: Application to visual servoing. *IEEE Trans. on Robotics and Automation*, 17(5):719–730, October 2001.

[7] G. Chesi, K. Hashimoto, D. Prattichizzo, and A. Vicino. A switching control law for keeping features in the field of view in eye-in-hand visual servoing. In *IEEE Int. Conf. on Robotics and Automation (ICRA'03)*, pages 3929–3934, Taipei, Taiwan, September 2003.

[8] S. Chiaverini. Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. *IEEE Trans. on Robotics and Automation*, 13(3):398–410, June 1997.

[9] N. Cowan, J. Weingarten, and D. Koditschek. Visual servoing via navigation functions. *IEEE Trans. on Robotics and Automation*, 18(4):521–533, August 2002.

[10] A. Deo and I. Walker. Robot subtask performance with singularity robustness using optimal damped least squares. In *IEEE Int. Conf. on Robotics and Automation (ICRA'92)*, pages 434–441, Nice, France, May 1992.

[11] B. Espiau, F. Chaumette, and P. Rives. A new approach to visual servoing in robotics. *IEEE Trans. on Robotics and Automation*, 8(3):313–326, June 1992.

[12] N. R. Gans and S. A. Hutchinson. An experimental study of hybrid switched approaches to visual servoing. In *IEEE Int. Conf. on Robotics and Automation (ICRA'03)*, pages 3061–3068, Taipei, Taiwan, September 2003.

[13] G. Hager. Human-machine cooperative manipulation with vision-based motion constraints. In *Workshop on visual servoing, IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'02)*, Lausane, Switzerland, October 2002.

[14] H. Hanafusa, T. Yoshikawa, and Y. Nakamura. Analysis and control of articulated robot with redundancy. In *IFAC, 8th Triennal World Congress*, volume 4, pages 1927–1932, Kyoto, Japan, 1981.

[15] S. Hutchinson, G. Hager, and P. Corke. A tutorial on visual servo control. *IEEE Trans. on Robotics and Automation*, 12(5):651–670, October 1996.

[16] M. Khatib and R. Chatila. An extended potentiel field approach for mobile robot sensor-based motions. In *Intelligent Autonomous Systems (IAS'4)*, pages 490–496, Karlsruhe, Germany, March 1995.

[17] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Int. Journal of Robotics Research*, 5(1):90–98, Spring 1986.

[18] C. Klein and S. Kittivatcharapong. Optimal force distribution for the legs of a walking machine with friction cone constraints. *IEEE Trans. on Robotics and Automation*, 6(1):73–85, February 1990.

[19] F. Lamiraux, D. Bonnafous, and O. Lefebvre. Reactive path deformation for nonholonomic mobile robots. *IEEE Trans. on Robotics*, 7(20):967–977, December 2004.

[20] S. LaValle and J. Kuffner. Randomized kinodynamic planning. *IEEE Trans. on Robotics and Automation*, 1:473–479, 1999.

[21] A. Liegeois. Automatic supervisory control of the configuration and behavior of multibody mechanisms. *IEEE Trans. on Systems, Man and Cybernetics*, 7(12):868–871, December 1977.

[22] A. Maciejewski and C. Klein. Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments. *Int. Journal of Robotics Research*, 4(3):109–117, Fall 1985.

[23] A. Maciejewski and C. Klein. Numerical filtering for the operation of robotic manipulators through kinematically singular configurations. *Journal of Robotic Systems*, 1988.

[24] E. Malis, F. Chaumette, and S. Boudet. 2 1/2 D visual servoing. *IEEE Trans. on Robotics and Automation*, 15(2):238–250, April 1999.

[25] N. Mansard and F. Chaumette. Tasks sequencing for visual servoing. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'04)*, pages 992–997, Sendai, Japan, November 2004.

[26] N. Mansard and F. Chaumette. Visual servoing sequencing able to avoid obstacles. In *IEEE Int. Conf. on Robotics and Automation (ICRA'05)*, pages 3154–3159, Barcelona, Spain, April 2005.

[27] G. Marani, Jinhyun Kim, Junku Yuh, and Wan Kyun Chung. A real-time approach for singularity avoidance in resolved motion rate control of robotic manipulators. In *IEEE Int. Conf. on Robotics and Automation (ICRA'02)*, pages 1973–1978, Washingtown DC, USA, May 2002.

[28] E. Marchand and G. Hager. Dynamic sensor planning in visual servoing. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'98)*, volume 3, pages 1988–1993, Leuven, Belgium, May 1998.

[29] Y. Mezouar and F. Chaumette. Path planning for robust image-based control. *IEEE Trans. on Robotics and Automation*, 18(4):534–549, August 2002.

[30] B. Nelson and P. Khosla. Strategies for increasing the tracking region of an eye-in-hand system by singularity and joint limits avoidance. *Int. Journal of Robotics Research*, 14(3):255–269, June 1995.

[31] L. Peterson, D. Austin, and D. Kragic. High-level control of a mobile manipulator for door opening. In *IEEE Int. Conf. on Robotics and Automation (ICRA'03)*, pages 2333–2338, Taipei, Taiwan, September 2003.

[32] R. Pissard-Gibolet and P. Rives. Applying visual servoing techniques to control a mobile hand-eye system. In *IEEE Int. Conf. on Robotics and Automation (ICRA'96)*, pages 166–171, Minneapolis, USA, May 1996.

[33] S. Quinlan and O. Khatib. Elastic bands: Connecting path planning and robot control. In *IEEE Int. Conf. on Robotics and Automation (ICRA'93)*, volume 2, pages 802–807, Atlanta, USA, May 1993.

[34] J. Rosen. The gradient projection method for nonlinear programmimg, part i, linear constraints. *SIAM Journal of Applied Mathematics*, 8(1):181–217, March 1960.

[35] C. Samson, M. Le Borgne, and B. Espiau. *Robot Control: the Task Function Approach*. Clarendon Press, Oxford, United Kingdom, 1991.

[36] L. Sentis and O. Khatib. Control of free-floating humanoid robots through task prioritization. In *IEEE Int. Conf. on Robotics and Automation (ICRA'05)*, pages 1718–1723, Barcelona, Spain, April 2005.

[37] B. Siciliano and J-J. Slotine. A general framework for managing multiple tasks in highly redundant robotic systems. In *IEEE Int. Conf. on Advanced Robotics (ICAR'91)*, pages 1211–1216, Pisa, Italy, June 1991.

[38] P. Soueres, V. Cadenat, and M. Djeddou. Dynamical sequence of multi-sensor based tasks for mobile robots navigation. In *7th Symp. on Robot Control (SYROCO'03)*, pages 423–428, Wroclaw, Poland, September 2003.

[39] P. Soueres, S. Tarbouriech, and B. Gao. A robust vision-based controller for mobile robots navigation: application to the task sequencing problem. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'05)*, Seville, Spain, December 2005.

[40] O. Tahri and F. Chaumette. Point-based and region-based image moments for visual servoing of planar objects. *IEEE Trans. on Robotics*, 21(6):1116–1127, December 2005.

[41] E. Yoshida. Humanoid motion planning using multi-level dof exploitation based on randomized method. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'05)*, pages 3378–3383, Edmonton, Canada, August 2005.