Enchaînement de tâches en asservissement visuel : application à l'évitement des butées articulaires

Visual servoing sequencing: application to joint limits avoidance

Nicolas Mansard François Chaumette

IRISA - ENS Cachan et INRIA Rennes
Campus de Beaulieu, 35042 Rennes, France
E-Mail : `Firstname.Lastname@irisa.fr`

Résumé

Les approches classiques en asservissement visuel contraignent dès le début de l'asservissement tous les degrés de liberté (ddl) du robot. Nous proposons dans cet article une nouvelle approche qui consiste à utiliser des tâches ne contraignant qu'un faible nombre de ddl tant que le robot est loin de son objectif, et à augmenter le nombre de contraintes uniquement quand le robot se rapproche de sa position finale. Tant que le robot est loin du but, le grand nombre de ddl restant permet d'éviter aisément les obstacles qui peuvent survenir. Plus proche de la position désirée, quand le nombre de contraintes imposées est plus grand, un controleur d'exécution peut retirer temporairement une des contraintes imposées pour dégager les ddl nécessaires à un évitement d'obstacle. Nous exposons une solution complète pour mettre cette idée générale en pratique. Cette solution est ensuite validée par des expérimentations sur un robot à six ddl.

Mots Clef

Asservissement visuel, enchaînement de tâches, redondance, évitement

Abstract

Classical visual servoing approaches tend to constrain all degrees of freedom (DOF) of the robot during the execution of a task. In this article a new approach is proposed. The key idea is to control the robot with a very under-constrained task when it is far from the desired position, and to incrementally constrain the global task by adding further tasks as the robot moves closer to the goal. As long as they are sufficient, the remaining DOF are used to avoid undesirable configurations, such as joint limits. Closer from the goal, when not enough DOF remain available for avoidance, an execution controller selects a task to be temporary removed from the applied tasks. The released DOF can then be used for the avoidance. A complete solution to implement this idea is proposed. Experiments that prove the validity of the approach are also provided.

Keywords

Visual servoing, tasks sequencing, redundancy, avoidance

1 Introduction

L'asservissement visuel propose des méthodes efficaces pour amener un robot jusqu'à une position désirée précise [10]. De plus, le contrôle est robuste aux bruits provenant du traitement d'image et aux erreurs de calibration du système robot - caméra. Toutefois, en contraignant tous les degrés de liberté (ddl) du robot pour accomplir la tâche de positionnement, ces méthodes sont incapables de prendre en compte les contraintes imposées par l'environnement, par exemple des obstacles, des occultations ou les butées du robot. Des approches plus récentes issues de la planification de mouvement [6, 14] permettent de prendre en compte ces contraintes. Mais cela impose au robot une trajectoire unique, ce qui nécessite une bonne connaissance de l'environnement. Nous pensons qu'il est possible de prendre en compte un environnement plus complexe, imposant de nombreuses contraintes, simplement en laissant au robot autant de ddl que possible : plutôt que de contrôler le robot dès le départ avec une tâche complète, contraignant tous les ddl, nous proposons dans cet article d'enchaîner des tâches incomplètes jusqu'à rejoindre la position désirée [16, 21, 17]. Tant que le robot est loin de sa position désirée, des tâches très simples sont appliquées, pour augmenter le nombre de trajectoires possibles. Les ddl restants peuvent être utilisés pour prendre en compte l'environnement, sans perturber l'exécution de la tâche appliquée [13]. Des tâches plus contraignantes sont progressivement ajoutées lorsque le robot se rapproche de sa position finale [12]. La prise en compte de l'environnement peut alors devenir problématique. Dans ce cas, des ddl supplémentaires sont dégagés en levant temporairement une des contraintes imposées au robot, choisie de manière adéquate.

De nombreuses trajectoires sont généralement possibles pour amener le robot de sa position initiale à la position désirée. Les méthodes classiques d'asservissement visuel choisissent en fait une de ces trajectoires, sans savoir si elle

est valide ou non. Par conséquent dans certains cas, cette trajectoire peut éventuellement amener le robot dans un minimum local, dans une singularité de la tâche ou du robot, ou encore dans un obstacle [2]. Pour obtenir une exécution parfaite, une première solution consiste à planifier le mouvement du robot pour choisir parmi toutes les trajectoires possibles celle qui est optimale au sens des critères choisis [6, 14]. Mais ce choix d'une trajectoire précise peut s'avérer difficile, en particulier si l'environnement est mal connu, ou s'il est changeant. Pourtant, si d'autres trajectoires sont possibles, et permettent de rejoindre aussi sûrement la position finale, pourquoi se limiter à une seule trajectoire ? Plutôt que de décider à l'avance de la trajectoire à suivre, les méthodes de *switching control* [8, 5] utilisent un ensemble de plusieurs lois de commande correspondant chacune à une trajectoire particulière, et passent de l'une à l'autre pour éviter les régions difficiles. Cette solution permet en principe d'augmenter la taille du domaine de stabilité, qui devient alors proche de l'union des domaines de stabilité de chaque loi de commande utilisée. De nombreux articles traitent de l'enchaînement de tâches, qui permet de rejoindre un but très éloigné de la position initiale en enchaînant de manière séquentielle une série de tâches généralement choisies à l'avance par le programmeur [16, 21, 17].

Notre approche se situe dans ce dernier groupe de travaux. La méthode proposée dans cet article permet de choisir en ligne la tâche à appliquer, de manière à rejoindre le but fixé tout en prenant en compte l'environnement. L'idée principale est de diviser une tâche d'asservissement visuel complète en plusieurs tâches ne contraignant pas tous les ddl du robot (des tâches *élémentaires*). Le robot se déplace dans le but de réaliser une de ces tâches élémentaires, tout en maintenant accomplies celles qu'il a déjà réalisées. A la fin de l'asservissement, le robot est complètement contraint par la somme des contraintes de toutes les tâches. Tant que suffisament de ddl sont disponibles, ceux-ci sont utilisés pour prendre en compte des contraintes supplémentaires, telles que l'évitement des butées du robot. Quand ces contraintes ne peuvent plus être assurées, à cause du nombre insuffisant de ddl restants, une tâche élémentaire déjà accomplie peut être retirée. Si elle est choisie de manière adéquate, cela permet de dégager les ddl nécessaires à l'évitement.

Dans la première partie de cet article, nous présenterons la structure de pile de tâches qui nous permet de déplacer le robot pour accomplir une tâche élémentaire tout en maintenant celles déjà réalisées à zéro. Une loi de commande est calculée directement d'après la pile de manière récursive, en utilisant le formalisme de la redondance [19]. Contrairement à la solution proposée dans [12], nous utiliserons directement une résolution des moindres carrés pour étendre ce formalisme à plus de deux tâches [20]. La loi de commande garantit de plus que la vitesse envoyée est continue, quelles que soient les manipulations effectuées sur la pile. Nous décrirons dans la seconde partie la méthode choisie pour utiliser les ddl restant pour prendre en compte des contraintes supplémentaires, comme par exemple les butées du robot. Nous utiliserons la méthode du gradient projeté [13, 11, 18]. Cette solution donne de bons résultats, tant que le nombre de ddl restant est grand. Quand la solution offerte par la méthode du gradient projeté n'est plus suffisante, en particulier quand le robot est trop contraint, la tâche élémentaire qui perturbe le plus l'exécution est automatiquement détéctée. Elle est alors temporairement retirée dans la pile, pour y être remise quand le problème est résolu. Une méthode générale permettant de réaliser cette idée est proposée dans la troisième partie. Enfin, nous terminerons en présentant les expériences réalisées sur un robot manipulateur.

2 La pile de tâches.

Dans cette section nous présentons la structure de pile de tâches qui permet d'enchaîner des tâches élémentaires tout en conservant accomplies les tâches déjà réalisées. La loi de commande est calculée d'après la pile de tâches, en respectant deux règles :

- toute tâche nouvellement ajoutée dans la pile ne doit pas perturber les tâches déjà présentes. En particulier, une tâche terminée doit rester à zéro ;
- la vitesse obtenue doit être continue, même quand une tâche est ajoutée ou retirée de la pile. Un saut de continuité signifierait une accélération infinie durant un court instant, ce qui impliquerait que le robot n'appliquerait pas correctement la commande imposée.

Le paragraphe 2.1 présente le formalisme de la redondance introduit dans [19]. Ce formalisme a été appliqué à l'asservisment visuel pour la première fois dans [7], et dans de nombreuses applications depuis (par exemple l'évitement d'occultation [13], ou la coopération homme-machine en robotique médicale [9]). L'idée générale est de n'utiliser que les ddl laissés libres par la tâche prioritaire pour réaliser au mieux la tâche secondaire. Le paragraphe 2.2 étend ce formalisme à un plus grand nombre de tâches. Le paragraphe 2.3 rappellera brièvement la méthode proposée dans [12] pour assurer la continuité de la commande.

2.1 Formalisme de la redondance pour deux tâches

Soit $\mathbf{q}$ le vecteur de position articulaire du robot. Soient $\mathbf{e_1}$ et $\mathbf{e_2}$ les deux tâches choisies, et $\mathbf{J_i} = \frac{\partial \mathbf{e_i}}{\partial \mathbf{q}}$ $(i = 1, 2)$ leur jacobien, définis par:

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

Le robot est contrôlé en fixant $\dot{\mathbf{q}}$. L'équation (1) doit donc être inversée. La solution générale (avec $i = 1$) est donnée par :

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

où $\mathbf{P_1}$ est la projection orthogonale sur le noyau de $\mathbf{J_1}$ et $\mathbf{J_1^+}$ est la pseudo-inverse (ou inverse au sens des moindres

carrées) de $\mathbf{J_1}$. $\mathbf{z}$ est arbitraire, et peut être utilisé pour appliquer une tâche secondaire, qui ne perturbera pas la tâche prioritaire $\mathbf{e_1}$. Nous utilisons $\mathbf{z}$ pour réaliser la tâche $\mathbf{e_2}$. En introduisant (2) dans (1) (avec $i = 2$), on obtient l'équation suivante, où $\mathbf{z}$ est l'inconnue :

$$\dot{\mathbf{e_2}} = \mathbf{J_2}\mathbf{J_1^+}\dot{\mathbf{e_1}} + \mathbf{J_2}\mathbf{P_1}\mathbf{z} \tag{3}$$

Une solution $\mathbf{z}$ peut être facilement calculée en inversant cette dernière équation. En introduisant cette solution dans (2), nous pouvons écrire :

$$\dot{\mathbf{q}} = \mathbf{J_1^+}\dot{\mathbf{e_1}} + \mathbf{P_1}(\mathbf{J_2}\mathbf{P_1})^+(\dot{\mathbf{e_2}} - \mathbf{J_2}\mathbf{J_1^+}\dot{\mathbf{e_1}}) \tag{4}$$

Comme $\mathbf{P_1}$ est hermitienne et idempotente (c'est une projection orthogonale), (4) s'écrit finalement :

$$\dot{\mathbf{q}} = \mathbf{J_1^+}\dot{\mathbf{e_1}} + \widetilde{\mathbf{J_2}}^+\widetilde{\dot{\mathbf{e_2}}} \tag{5}$$

avec $\widetilde{\mathbf{J_2}} = \mathbf{J_2}\mathbf{P_1}$ et $\widetilde{\dot{\mathbf{e_2}}} = \dot{\mathbf{e_2}} - \mathbf{J_2}\mathbf{J_1^+}\dot{\mathbf{e_1}}$. $\widetilde{\mathbf{J_2}}$ est le jacobien limité de la tâche $\mathbf{e_2}$. Il nous donne l'espace de travail de la tâche $\mathbf{e_2}$ ne perturbant pas la tâche $\mathbf{e_1}$. $\widetilde{\dot{\mathbf{e_2}}}$ correspond à la seconde fonction de tâche $\dot{\mathbf{e_2}}$ de laquelle on a soustrait la partie de la commande $\mathbf{J_2}\mathbf{J_1^+}\dot{\mathbf{e_1}}$ déjà réalisée par $\mathbf{e_1}$. Une explication plus intuitive de ces équations peut être trouvée dans [1].

2.2 Extension du formalisme de la redondance à plus de deux tâches

Soient $(\mathbf{e_1}, \mathbf{J_1})$ ... $(\mathbf{e_n}, \mathbf{J_n})$ n tâches. Nous voulons étendre (5) à ces n tâches. La tâche $\mathbf{e_i}$ ne doit pas être perturbée par la tâche $\mathbf{e_j}$ si $i < j$. Une méthode récursive pour étendre cette équation est proposée dans [20] :

$$\dot{\mathbf{q_i}} = \dot{\mathbf{q}}_{\mathbf{i-1}} + (\mathbf{J_i}\mathbf{P^A_{i-1}})^+(\dot{\mathbf{e_i}} - \mathbf{J_i}\dot{\mathbf{q}}_{\mathbf{i-1}}) \tag{6}$$

où $\mathbf{P^A_i}$ est la projection orthogonale sur le noyau du jacobien augmenté $\mathbf{J^A_i}$

$$\mathbf{J^A_i} = \begin{bmatrix} \mathbf{J_1} \\ \vdots \\ \mathbf{J_i} \end{bmatrix} \tag{7}$$

La récurrence est initialisée avec $\dot{\mathbf{q}}_{\mathbf{0}} = 0$. La vitesse articulaire prenant en compte toute la pile est finalement $\dot{\mathbf{q}} = \dot{\mathbf{q}}_{\mathbf{n}}$. En utilisant directement cette équation, la projection $\mathbf{P^A_i}$ doit être calculée complètement à chaque étape de la récurrence. Une méthode pour calculer récursivement cette projection est proposée dans [1] :

$$\mathbf{P^A_i} = \mathbf{P^A_{i-1}} - \widetilde{\mathbf{J_i}}^+\widetilde{\mathbf{J_i}} \tag{8}$$

où $\widetilde{\mathbf{J_i}} = \mathbf{J_i}\mathbf{P^A_{i-1}}$ est le jacobien limité de la tâche $\mathbf{e_i}$. La récurrence est initialisée par $\mathbf{P^A_0} = \mathbf{I}$ (la matrice identité).

2.3 Continuité de la loi de commande

La loi de commande est généralement obtenue en imposant une décroissance exponentielle de l'erreur :

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

Comme $\dot{\mathbf{e}} = \mathbf{J}\dot{\mathbf{q}}$, on obtient alors la loi de commande suivante :

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

où $\widehat{\mathbf{J^+}}$ est une approximation de la pseudo-inverse de $\mathbf{J}$ et λ est un paramètre qui permet d'influencer la vitesse du robot. La fonction f_1 est choisie par le programmeur pour relier $\dot{\mathbf{e}}$ à $\mathbf{e}$. On choisit généralement $f_1(\mathbf{e}) = -\lambda\mathbf{e}$ pour imposer une décroissance exponentielle de l'erreur.
Les problèmes de continuité interviennent lors du changement de tâche $\mathbf{e}$. Ils sont dus au manque de contraintes initiales sur la valeur de $\dot{\mathbf{e}}$. Soit $\mathbf{e_A}$ une tâche complète, utilisée pour contrôler le robot jusqu'au temps $t = 0$. A cet instant, la loi de commande change pour utiliser une seconde tâche $\mathbf{e_B}$. Comme $\mathbf{e}$ et $\dot{\mathbf{q}}$ sont liées linéairement, la continuité de $\dot{\mathbf{q}}$ ne peut pas être assurée au temps $t = 0$. Soueres et al. ont proposé une solution à ce problème dans [21]. Ils utilisent une équation linéaire du second ordre plutôt que (10) pour prendre en compte deux contraintes intiales $(\mathbf{e}(0), \dot{\mathbf{e}}(0))$:

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

Les deux paramètres α et β sont utilisés pour fixer à la fois la vitesse de convergence de l'erreur et la durée de la période de transition entre les deux lois de commande. Le problème principal de cette méthode est le couplage entre ces deux paramètres.
Dans [12], nous avons proposé d'utiliser plutôt une équation différencielle du premier ordre non homogène pour assurer en même temps la continuité et un bon découplage des deux paramètres de contrôle. L'équation choisie est :

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

où $\rho(t)$ est la partie non homogène de l'équation :

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

En introduisant (12) dans (2), on obtient finalement la loi de commande générale assurant la continuité après l'instant de changement de tâche :

$$\dot{\mathbf{q}} = -\lambda\mathbf{J^+}\mathbf{e} + e^{-\mu \cdot t}\mathbf{K} \tag{14}$$

où $\mathbf{K} = \mathbf{J^+}(\dot{\mathbf{e_A}}(0) + \lambda\mathbf{e_B}(0))$. Cette loi de commande est composée de deux parties. La première, paramétrée par λ, assure la décroissance de l'erreur. La seconde, paramétrée par μ, assure la continuité pendant une courte période temporelle. Les deux gains λ et μ sont donc complétement découplés. Pour assurer que la période de transition soit plus courte que la période de convergence, il suffit de choisir μ supérieur à λ (nous utilisons en pratique $\mu = 10\lambda$).

2.4 Loi de commande

Soient $(\mathbf{e_1}, \ldots, \mathbf{e_n})$ une pile de n tâches. La vitesse de décroissance de chaque tâche est fixée par :

$$\dot{\mathbf{e}} = \begin{bmatrix} \dot{\mathbf{e_1}} \\ \vdots \\ \dot{\mathbf{e_n}} \end{bmatrix} = - \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{bmatrix} \begin{bmatrix} \mathbf{e_1} \\ \vdots \\ \mathbf{e_n} \end{bmatrix} = -\Lambda \mathbf{e} \tag{15}$$

L'équation (4) peut alors être réécrite $\dot{\mathbf{q}} = \mathbf{L_p}\dot{\mathbf{e}}$, (l'expression explicite de $\mathbf{L_p}$ est laissée au lecteur). En utilisant (12) et (15) on déduit finalement l'expression complète de la loi de commande calculée à partir d'une pile de tâches :

$$\begin{cases} \dot{\mathbf{q}}_\mathbf{i} = \dot{\mathbf{q}}_\mathbf{i-1} + (\mathbf{J_i}\mathbf{P}^\mathbf{A}_\mathbf{i-1})^+(-\lambda_i \mathbf{e_i} - \mathbf{J_i}\dot{\mathbf{q}}_\mathbf{i-1}) \\ \dot{\mathbf{q}} = \dot{\mathbf{q}}_\mathbf{n} + e^{-\mu\cdot(t-\tau)} \cdot \left(\dot{\mathbf{e}}(\tau) + \Lambda \mathbf{e}(\tau)\right) \end{cases} \tag{16}$$

où τ est l'instant de la dernière modification de la pile et où $\dot{\mathbf{e}}(\tau)$ est calculé en fonction de $\dot{\mathbf{q}}$ en utilisant (1) ($\dot{\mathbf{e}}(\tau) = \mathbf{J}\dot{\mathbf{q}}(\tau)$).

3 Evitement d'obstacle simple utilisant la méthode du gradient projeté

Dans la partie précédente, nous avons proposé une loi de commande calculée à partir d'une pile de tâches, qui permet au robot de réaliser une tâche complète, par exemple une tâche de positionnement. Cette loi de commande est déduite uniquement des fonctions de tâche élémentaire $\mathbf{e_i}$. Dans notre cas, celles-ci sont basées sur la décroissance d'erreurs entre primitives visuelles. Pour intégrer un tel asservissement dans un système robotique complexe, la loi de commande doit aussi assurer que le robot va éviter les configurations impossibles (par exemple les butées articulaires, les occultations, les obstacles ou encore les singularitées articulaires). Une première solution à ce problème a été proposée dans [15]. Elle consiste à minimiser une fonction (appelée *objective function*) qui réalise un compromis entre la tâche d'asservissement visuel et l'évitement des configurations problématiques. Une deuxième méthode, plus proche des travaux présentés dans le paragraphe 2, est utilisée dans [13]. Cette solution, appelée méthode du gradient projeté, est basée sur le formalisme de la fonction de tâche [19]. On construit une fonction de coût atteignant son maximum sur les situations à éviter. Cette fonction de coût est ensuite intégrée dans une tâche secondaire, dont seules les composantes ne perturbant pas la tâche prioritaire sont prises en compte. Ceci est réalisé en projetant cette tâche secondaire sur le noyau de la tâche principale.

Nous utiliserons dans la suite le terme générique d'obstacle pour désigner toutes les situations à éviter. Les expériences présentées dans le paragraphe 5 ont été réalisées avec un évitement de butées, mais la méthode du gradient peut s'appliquer à des obstacles de nature diverse [13, 4]. Nous rappellerons tout d'abord rapidement comment calculer une loi de commande utilisant le gradient d'une fonction de coût [7]. Puis nous calculerons la loi de commande complète, basée sur la pile de tâches et prenant en compte les obstacles. Nous présenterons alors rapidement les problèmes sur lesquels débouche une telle méthode, ce qui permettra d'introduire la dernière partie de l'article.

3.1 Méthode générale du gradient projeté

Dans cette approche, le robot se déplace selon un potentie répulsif V qui l'éloigne des obstacles. Considérons le problème suivant :

$$\min V(\mathbf{q}), \quad \mathbf{q} \in \mathbf{R}^k \tag{17}$$

où k est le nombre d'articulations du robot. La solution classique consiste à déplacer le robot en suivant le gradient de la fonction de potentiel calculé dans l'espace articulaire.

$$\dot{\mathbf{q}} = -\kappa \mathbf{g}(\mathbf{q}) = -\kappa \vec{\nabla}^\top_\mathbf{q} V \tag{18}$$

où κ est un scalaire positif, utilisé comme un gain. Cette tâche secondaire (18) est alors projeté sur le noyau de la tâche prioritaire, en utilisant (2).

3.2 Méthode du gradient projeté et pile de tâches

Nous voulons utiliser le gradient de la fonction de coût comme la dernière tâche de la pile. La loi de commande (18) est donc projetée sur le noyau de chaque tâche de la pile. En utilisant les notations de (16), la loi de commande complète est finalement :

$$\dot{\mathbf{q}} = \dot{\mathbf{q}}_\mathbf{n} + e^{-\mu\cdot(t-\tau)} \cdot \left(\dot{\mathbf{e}}(\tau) + \Lambda \mathbf{e}(\tau)\right) - \kappa \mathbf{P}^\mathbf{A}_\mathbf{n} \mathbf{g} \tag{19}$$

On voit dans cette équation que l'importance de l'évitement dépend de deux facteurs. Tout d'abord, il dépend de la projection orthogonale $\mathbf{P}^\mathbf{A}_\mathbf{n}$. Quand la pile est presque vide, le rang de $\mathbf{P}^\mathbf{A}_\mathbf{n}$ est élevé, et la loi d'évitement n'est pratiquement pas modifiée. Par contre, quand la pile est pratiquement pleine, le rang de la matrice $\mathbf{P}^\mathbf{A}_\mathbf{n}$ est faible. La loi d'évitement est alors fortement perturbée par la projection. Ceci est particulièrement vrai lorsque la direction privilégiée du gradient est comprise dans l'image vectorielle de la projection. Le gradient projeté est évidement 0 quand la pile est pleine. Il n'est alors plus pris du tout en compte, et plus rien n'est fait pour éviter les obstacles. Le second facteur est le gain κ, qui définit *a priori* l'importance de l'évitement dans la loi de commande finale. Le choix de ce paramètre est très important. Si κ est trop petit, la force répulsive sera trop faible pour éloigner le robot des obstacles. Au contraire, si κ est trop élevé, la loi de commande risque de saturer ou d'osciller. Des méthodes existent pour régler automatiquement ce gain (par exemple [4] pour l'évitement de butées). Toutefois, il est difficile de généraliser ces méthodes pour prendre en compte en même temps de nombreux obstacles de nature différente. De plus, ces méthodes ne résolvent pas le problème posé par le rang de $\mathbf{P}^\mathbf{A}_\mathbf{n}$.

Il nous a plutôt semblé préférable de modifier directement la pile de tâches pour modifier le comportement du robot lorsque la méthode du gradient projeté n'est pas suffisante. Le gradient projeté est utilisé pour garder le robot hors des obstacles tant que cela est possible. Quand la méthode n'est plus suffisante, une solution plus efficace consiste à détecter quelle tâche de la pile est responsable de la partie de la loi de commande qui envoye le robot dans l'obstacle, et à retirer cette tâche de la pile. Cette solution est détaillée dans la partie suivante.

4 Construction d'un contrôleur de pile

Nous présentons dans cette section le contrôleur de pile qui permet de réaliser l'idée présentée à la fin du paragraphe précédent. Deux actions sont réalisables sur la pile : ajouter une tâche ou retirer une tâche. Une tâche est ajoutée dans la pile lorsque la vitesse envoyée au robot passe sous un seuil fixé, c'est-à-dire quand toutes les tâches de la pile sont réalisées au mieux, selon leur ordre dans la pile. L'ordre d'ajout des tâches est fixé *a priori*. Choisir l'ordre d'ajout des tâches *en ligne* est un problème difficile, qui fait partie de nos perspectives de recherche importantes. A l'inverse, une tâche doit être retirée de la pile quand la loi de commande envoie le robot dans l'obstacle. Deux critères doivent donc être choisis, le premier pour détecter à quel moment une tâche doit être retirée de la pile, le second pour choisir quelle tâche enlever.

4.1 A quel moment retirer une tâche?

Le critère consiste simplement en une étape de prédiction réalisée avant d'envoyer la commande au robot. Notons $\mathbf{q}(t)$ la position articulaire courante du robot. La position prédite $\widehat{\mathbf{q}}(t+1)$ est simplement :

$$\widehat{\mathbf{q}}(t+1) = \mathbf{q}(t) + \Delta t \dot{\mathbf{q}} \tag{20}$$

où $\dot{\mathbf{q}}$ et la loi de commande, calculée en utilisant (19). Une tâche doit être retirée si $V(\widehat{\mathbf{q}}(t+1))$ est au dessus d'un certain seuil.

4.2 Quelle tâche retirer ?

L'idée est de détecter quelle tâche entre en conflit avec le gradient d'évitement d'obstacle. Nous proposons deux critères qui peuvent être calculés pour chaque tâche de la pile. La tâche à retirer est celle qui correspond à la valeur optimale des critères calculés. Nous utilisons simultanément deux critères pour rendre le choix plus robuste. Dans la suite, nous calculons les deux critères pour une tâche $\mathbf{e_i}$, dont le jacobien est $\mathbf{J_i}$, et pour un gradient d'évitement $\mathbf{g}(\mathbf{q})$.

Premier critère. Le premier critère compare directement les directions vectorielles du gradient et de la commande provenant de la tâche $\mathbf{e_i}$. La tâche à retirer sera celle dont la direction vectorielle correspond le plus à l'opposé de celle du gradient. Un exemple pratique du calcul de ce critère est donné par la figure 1.

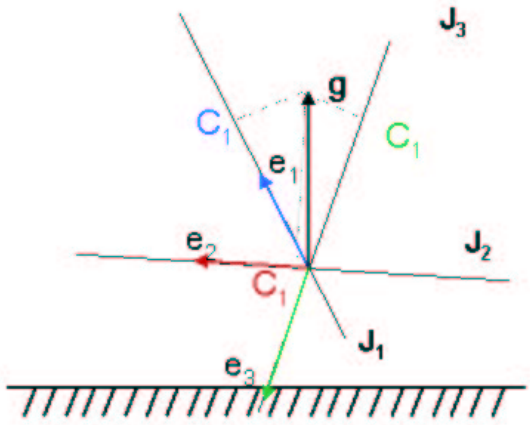

FIG. 1 – Calcul du critère $\mathcal{C}_1$. Trois tâches sont présentes dans la pile. Le robot est situé au point d'application des trois tâches. Un obstacle (représenté par le trait hachuré) est proche du robot. Le gradient d'évitement de cet obstacle est le vecteur $\mathbf{g}$. On voit intuitivement que la tâche qui conduit le robot sur l'obstacle est la tâche $\mathbf{e_3}$. Le critère maximal est bien $\mathcal{C}_1(\mathbf{e_3})$, représenté en vert ($\mathcal{C}_1(\mathbf{e_1})$ est négatif, et $\mathcal{C}_1(\mathbf{e_2})$ est presque nul).

Le critère est donc calculé en effectuant le produit scalaire des deux vecteurs projetés dans le même espace. Le choix le plus logique pour cet espace commun est l'espace de commande, c'est-à-dire l'espace articulaire. Le critère $\mathcal{C}_1$ est donc donné par :

$$\mathcal{C}_1 = - < \mathbf{J_i^+ e_i} | \mathbf{g} > \tag{21}$$

Un autre espace commun peut être utilisé, par exemple celui de la tâche, en utilisant $\mathcal{C}_{1\mathbf{b}} =< \mathbf{e_i} | \mathbf{J_i g} >$. Dans ce cas, le produit scalaire est réalisé dans un espace différent pour chaque tâche. Les expériences nous ont montré que le comportement du contrôleur de pile était très semblable pour ces deux critères.

Le premier critère dépend donc linéairement de la fonction de tâche $\mathbf{e_i}$. Si la tâche est pratiquement réalisée ($\mathbf{e_i}$ est faible), le critère sera faible également. Expérimentalement, nous avons remarqué que le contrôleur utilisant (21) enlève pratiquement toujours la dernière tâche ajoutée. Nous avons donc choisi de normaliser ce critère :

$$\mathcal{C}_1{}' = \frac{1}{||\mathbf{e_i}||} \mathcal{C}_1 \tag{22}$$

En utilisant cette définition, le choix est basé uniquement sur la direction vectorielle et non plus sur la norme. Toutefois, si la tâche est presque accomplie, la normalisation revient à une division par une valeur proche de zéro, ce qui peut amener des résultats instables. Ce problème est résolu par le second critère.

Second critère. Le gradient est projeté sur le noyau de chaque tâche lors du calcul de la loi de commande finale. Le second critère estime donc la contribution de chaque tâche à cette projection. L'idée est cette fois de retirer la tâche dont la contribution perturbe le plus le gradient d'évitement.

$$\mathcal{C}_2 = ||\mathbf{P_i g}|| \tag{23}$$

où $\mathbf{P_i} = \mathbf{I} - \mathbf{J_i^+ J_i}$ est la projection orthogonale sur l'espace nul de la tâche. Comme $\mathbf{P_i}$ est une projection, on a pour tout vecteur $\mathbf{x}$, $||\mathbf{P_i x}|| \leq ||\mathbf{x}||$. Plus le gradient est inclu dans l'espace nul, moins il sera perturbé par la projection, et plus le critère sera fort. Il faut donc retirer de la pile la tâche correspondant au minimum de $\mathcal{C}_2$. La figure 2 explique de manière plus intuitive l'idée du critère $\mathcal{C}_2$ par un exemple.

Une autre manière de calculer $\mathcal{C}_2$. Nous présentons dans ce paragraphe une seconde manière de calculer $\mathcal{C}_2$. L'idée est cette fois de vérifier si le gradient est présent dans le sous espace des commandes possibles engendrées par la tâche. Ce sous espace est donné par (2) : c'est l'image de $\mathbf{J_i^+}$. Soit $(\mathbf{v_1} \ldots \mathbf{v_k})$ une base de l'image de $\mathbf{J_i^+}$ (où k est le rang de $\mathbf{J_i^+}$). Le critère est alors la norme du gradient projeté dans l'image de $\mathbf{J_i^+}$

$$\mathcal{C}_{2\mathbf{b}} = ||\sum_{i=1}^{k} (\mathbf{g} \cdot \mathbf{v_i})\mathbf{v_i}|| \tag{24}$$

Prouvons que $\mathcal{C}_2$ et $\mathcal{C}_{2\mathbf{b}}$ sont équivalents. La projection orthogonale est indépendante de la base de l'image de $\mathbf{J_i^+}$. On peut donc choisir une base quelconque, par exemple la base $\mathbf{V}$ issue de la décomposition en valeurs singulières (SVD) de $\mathbf{J_i}$:

$$\mathbf{J_i} = \mathbf{USV}^T \tag{25}$$

où $\mathbf{V}^T$ est la transposée de $\mathbf{V}$. Considérons que les valeurs singulière sont ordonnées telles que $\mathbf{V} = (\mathbf{V_0 V_1})$ où les vecteurs de $\mathbf{V_0}$ (respectivement de $\mathbf{V_1}$) correspondent aux valeurs singulières nulles (respectivement non nulles). Le critère $\mathcal{C}_{2\mathbf{b}}$ s'écrit alors :

$$\mathcal{C}_{2\mathbf{b}} = ||\mathbf{V_1 V_1}^T \mathbf{g}|| \tag{26}$$

En utilisant la SVD, (23) s'écrit :

$$\mathcal{C}_2 = ||\mathbf{J_i^+ J_i g}|| = ||\mathbf{V_0 V_0}^T \mathbf{g}|| = ||(\mathbf{I} - \mathbf{V_1 V_1}^T)\mathbf{g}|| \tag{27}$$

$\mathcal{C}_2$ est minimum quand $\mathcal{C}_{2\mathbf{b}}$ est maximum. En fait, $\mathcal{C}_2$ vérifie si le gradient n'est pas dans le noyau du jacobien, alors que $\mathcal{C}_{2\mathbf{b}}$ vérifie si il est dans son image, ce qui est équivalent. Les expériences ont validées cette équivalence. Dans la suite, nous ne considérerons donc que $\mathcal{C}_2$.

5 Expérimentations et résultats

Nous présentons dans cette section les expérimentations qui valident les méthodes que nous avons exposées plus haut. Ces expériences ont été effectuées sur un robot à six ddl, en utilisant une caméra embarquée. La caméra doit se positionner par rapport à une cible simple, un rectangle composé de quatre points blancs aisément détectables. En facilitant le traitement d'image, nous avons pu valider plus spécifiquement la partie contrôle, qui nous intéressait plus particulièrement dans ce travail. Le robot devait de plus éviter ses butées articulaires lors du positionnement.

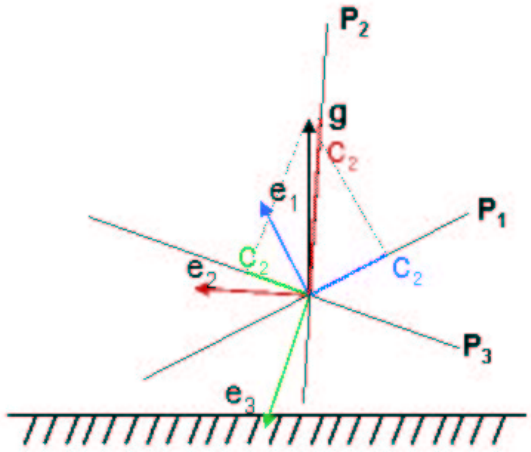

FIG. 2 – Calcul du critère $\mathcal{C}_2$. La situation est la même que sur la figure 1. Les projecteurs $\mathbf{P_1}$, $\mathbf{P_2}$ et $\mathbf{P_3}$ sont représentés par leur direction vectorielle. Ces droites sont orthogonales aux vecteurs $\mathbf{e_1}$, $\mathbf{e_2}$ et $\mathbf{e_3}$ correspondants. Le gradient d'évitement est projeté sur chacune de ces droites. Le critère minimal est bien $\mathcal{C}_2(\mathbf{e_3})$, représenté en vert. Dans le cas du critère $\mathcal{C}_2$, le signe de la tâche n'est pas pris en compte. La valeur du critère $\mathcal{C}_2(\mathbf{e_1})$ est très proche de $\mathcal{C}_2(\mathbf{e_3})$ ($\mathbf{e_1}$ empêche en effet l'évitement, même si elle ne conduit pas elle même le robot dans l'obstacle).

Les expériences exigeaient des déplacements importants, passant par une large partie du domaine articulaire. Pour de tels déplacements, les algorithmes classiques d'asservissement visuel ont échoué, soit à cause de l'importante erreur initiale, soit à cause de la proximité des butées. L'asservissement utilisant une pile de tâches (19) et un contrôleur de pile (20) réussit par contre à mener le robot jusqu'à la position désirée. L'exécution que nous présentons ici emmène le robot d'un coin du domaine articulaire à l'autre. Les images initiales et finales sont présentées Fig. 4. A la position désirée, la cible est centrée mais n'est pas parallèle au plan de l'image (l'angle est de 20dg). Le déplacement demandé est ($t_x = -491$mm, $t_y = -120$mm, $t_z = 340$mm, $(u\theta)_x = -70$dg, $(u\theta)_y = -35$dg, $(u\theta)_z = 110$dg).

Nous présentons tout d'abord les quatre tâches d'asservissement visuel utilisées pour remplir la pile (paragraphe 5.1) ainsi que la loi d'évitement utilisée lors des expériences (paragraphe 5.2). Les quatre tâches choisies utilisent respectivement le centre de gravité de l'objet, l'angle d'une de ses diagonales, et les moments de seconds et de troisième ordre. Nous présentons alors dans le pa-

ragraphe 5.3 deux exécutions réalisées avec la méthode incomplète, la première sans loi d'évitement, la seconde avec uniquement l'évitement simple, qui échouent à positionner le robot. L'exécution utilisant l'intégralité de la méthode proposée, qui réussi à amener le robot à la position désirée, sera finalement détaillée.

5.1 Quatre tâches contraignant six ddl

Les quatre fonctions de tâche $\mathbf{e_i}$ utilisées dans la suite sont calculées d'après l'erreur calculée dans l'image selon la méthode introduite dans [7] :

$$\mathbf{e_i} = \mathbf{s_i} - \mathbf{s_i^*} \tag{28}$$

où $\mathbf{s_i}$ est la valeur des primitives visuelles calculées dans l'image courante et $\mathbf{s_i^*}$ les valeurs calculées dans l'image désirée. La matrice d'interaction $\mathbf{L_{s_i}}$ correspondant à $\mathbf{s_i}$ est définie par l'égalité $\dot{\mathbf{s_i}} = \mathbf{L_{s_i}}\mathbf{v}$, où $\mathbf{v}$ est le torseur cinématique de la caméra. D'après (28), on déduit immédiatement le lien entre la matrice d'interaction $\mathbf{L_{s_i}}$ et le jacobien de la tâche $\mathbf{J_i}$:

$$\mathbf{J_i} = \mathbf{L_{s_i}}\mathbf{M}\mathbf{J_q} \tag{29}$$

où $\mathbf{J_q}$ est le jacobien articulaire du robot ($\dot{\mathbf{r}} = \mathbf{J_q}\dot{\mathbf{q}}$) et $\mathbf{M}$ est une matrice reliant la vitesse de la caméra $\mathbf{v}$ à la variation de la position de la caméra $\mathbf{r}$ dans l'espace de paramètres choisis ($\mathbf{v} = \mathbf{M}\dot{\mathbf{r}}$). Hors des singularités du robot, le jacobien articulaire est de rang plein, et les deux matrices $\mathbf{J_i}$ et $\mathbf{L_{s_i}}$ ont la même image et le même rang. Calculer les opérateurs de projection de (2) en utilisant $\mathbf{J_i}$ permet donc de garantir l'invariance de $\mathbf{s_i}$.

Pour chaque tâche, nous donnons la valeur des primitives visuelles et la matrice d'interaction associée, calculées à partir de l'image courante. On note $p_i = (x_i, y_i)$ la position des quatre points dans l'image ($i \in [1 \ldots 4]$). On note $P_i = (X_i, Y_i, Z_i)$ la position de ces points dans le repère de la caméra.

La première tâche $\mathbf{e_{grav}}$ est basée sur la position du centre de gravité de l'objet. Les primitives associées sont simples à calculer :

$$\begin{bmatrix} x_g \\ y_g \end{bmatrix} = \begin{bmatrix} \frac{1}{4} \cdot \sum_{i=1}^{4} x_i \\ \frac{1}{4} \cdot \sum_{i=1}^{4} y_i \end{bmatrix} \tag{30}$$

Comme la projection perspective ne conserve pas les barycentres, le point g dans l'image ne correspond pas à un point physique en 3-D. Nous avons malgré tout choisi de faire cette approximation, et de considérer g comme l'image du centre de gravité de notre objet 3-D. La matrice d'interaction approximée est [7] :

$$\widehat{\mathbf{L}}_\mathbf{g} = \begin{bmatrix} -\frac{1}{Z_g} & 0 & \frac{x_g}{Z_g} & x_g y_g & -1 - x_g^2 & y_g \\ 0 & \frac{1}{Z_g} & \frac{y_g}{Z_g} & 1 + y_g^2 & -x_g y_g & -x_g \end{bmatrix} \tag{31}$$

où Z_g est la profondeur du point qui correspondrait au centre de gravité dans l'image. On peut considérer que c'est la profondeur du centre de gravité de l'objet. Z_g n'est pas connu, et doit donc être approximée par une valeur *a priori*.

La seconde tâche $\mathbf{e}_\alpha$ fait principalement exécuter à la caméra un mouvement de rotation autour de l'axe optique, de manière à orienter correctement l'objet dans l'image. La primitive visuelle utilisée est l'angle α d'un segment dans l'image, c'est-à-dire ici une des diagonales du rectangle. La matrice d'interaction approchée est [7] :

$$\widehat{\mathbf{L}}_\alpha = \begin{bmatrix} 0 \\ 0 \\ 0 \\ x_c \sin^2\alpha + y_c \cos\alpha \sin\alpha \\ x_c \cos\alpha \sin\alpha - y_c \cos^2\alpha \\ -1 \end{bmatrix}^T \tag{32}$$

où (x_c, y_c) est le centre du segment.

La troisième tâche $\mathbf{e_Z}$ utilise les moments centrés de l'objet pour contrôler principalement la distance entre la caméra et la cible. La primitive visuelle la plus intuitive pour réaliser cette tâche est sans doute l'aire de l'objet, c'est-à-dire le moment d'ordre zéro d'un objet. Puisque nous utilisons un "objet" constitué de points, les moments centrés d'ordre deux, qui permettent de réaliser un déplacement équivalent, sont un choix plus logique. Les moments sont calculés en utilisant les travaux présentés dans [22]. Les moments centrés $\mu_{i,j}$ d'un objet composé de N points sont définis par :

$$\mu_{i,j} = \sum_{k=1}^{N} (x_k - x_g)^i \cdot (y_k - y_g)^j \tag{33}$$

Nous utilisons la primitive visuelle a_n calculée à partir de μ_{20} et μ_{02} :

$$a_n = Z^* \cdot \sqrt{\frac{a^*}{a}} \tag{34}$$

où $a = \mu_{20} + \mu_{02}$. La matrice d'interaction associée peut également être trouvée dans [22]. Quand l'objet est parallèle au plan de l'image, elle a la forme suivante :

$$\widehat{\mathbf{L}}_\mathbf{Z} = \begin{bmatrix} 0 & 0 & -1 & -\epsilon_1 & \epsilon_2 & 0 \end{bmatrix} \tag{35}$$

avec

$$\begin{cases} \epsilon_1 = y_g + (y_g \cdot \mu_{02} + x_g \cdot \mu_{11})/a \\ \epsilon_2 = x_g + (x_g \cdot \mu_{20} + y_g \cdot \mu_{11})/a \end{cases}$$

La dernière tâche $\mathbf{e_R}$ provient des travaux décrits dans [3]. Elle utilise les moments d'ordre trois pour découpler υ_x de ω_y et υ_y de ω_x. L'effet visuel correspond à une correction de l'effet de perspective. Les primitives visuelles sont données par :

$$\begin{cases} \mathbf{s_x} = \sqrt{a}(s_1 t_1 + s_2 t_2)/K \\ \mathbf{s_y} = \sqrt{a}(s_2 t_1 - s_1 t_2)/K \end{cases} \tag{36}$$

avec

$$\begin{cases} s_1 = \mu_{03} - 3\mu_{21}, t_1 = (\mu_{20} - \mu_{02})^2 - 4\mu_{11}^2 \\ s_2 = \mu_{30} - 3\mu_{12}, t_2 = \mu_{11}(\mu_{20} - \mu_{02}) \\ K = (\mu_{20} - \mu_{02})^{\frac{3}{2}}\left((\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2\right) \end{cases}$$

La matrice d'interaction correspondante peut être trouvée dans [3].

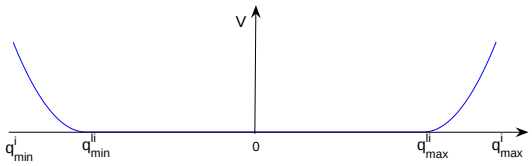

FIG. 3 – Fonction de coût associée à l'évitement de butées articulaires

5.2 Evitement de butées

Les expériences utilisent un évitement de butées articulaires comme exemple de gradient projeté. La fonction de coût associée est définie directement dans l'espace articulaire. Elle atteint sa valeur maximale quand le robot atteint une de ses butées, et vaut zéro quand le robot est loin de ses limites.
On note $\bar{q}_i^{\min}$ et $\bar{q}_i^{\max}$ les bornes inférieure et supérieure de l'axe i. La position $\mathbf{q}$ du robot est acceptable si pour tout i, $\mathbf{q}_i \in [\bar{q}_{\ell i}^{\min}, \bar{q}_{\ell i}^{\max}]$, où

$$\begin{cases} \bar{q}_{\ell i}^{\min} = \bar{q}_i^{\min} + \rho \bar{q}_i \\ \bar{q}_{\ell i}^{\max} = \bar{q}_i^{\max} - \rho \bar{q}_i \end{cases} \tag{37}$$

avec $\bar{q}_i = \bar{q}_i^{\max} - \bar{q}_i^{\min}$ la largeur de l'intervalle de chaque articulation et ρ un paramètre fixé par l'utilisateur dans l'intervalle $[0, 1/2]$ (typiquement $\rho = 0.1$). $\bar{q}_{\ell i}^{\min}$ et $\bar{q}_{\ell i}^{\max}$ sont les seuils d'activation de l'évitement de butées. Tant que le robot reste dans l'intervalle acceptable, la commande d'évitement est nulle. La fonction de coût provient de [4] (cf. Fig. 3) :

$$V(\mathbf{q}) = \frac{1}{2} \sum_{i=1}^{n} \frac{{\delta_i}^2}{\Delta \bar{q}_i} \tag{38}$$

avec

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

5.3 Résultats

Les premières expériences ont vérifié que les asservissements visuels classiques échouaient à réaliser la tâche demandée. En particulier, en empilant les quatre tâches en une seule tâche complète, on conduit le robot dans la butée correspondant au mouvement de recul selon l'axe optique. Nous avons alors réalisé une exécution utilisant la pile de tâches sans aucun évitement d'obstacle (16). Les positions articulaires du robot sont données sur la figure 5. Le robot atteint la butée correspondant au mouvement selon l'axe optique en reculant la caméra pour accomplir la troisième tâche. En ajoutant l´évitement d'obstacle simple (19), le robot peut utiliser les ddl restant pour éviter la butée lors de l'exécution de cette tâche (Fig. 6). On voit que la caméra se déplace selon ses axes x et y (articulations 1 et 2) pour pouvoir reculer sans toucher sa butée. Mais l'exécution échoue lors de la quatrième tâche. A ce moment, la pile est pleine, plus aucun ddl n'est disponible pour l'évitement. La quatrième tâche amène le robot à réaliser un mouvement de rotation centré sur l'objet, ce qui l'amène dans la butée à l'arrière de la caméra.

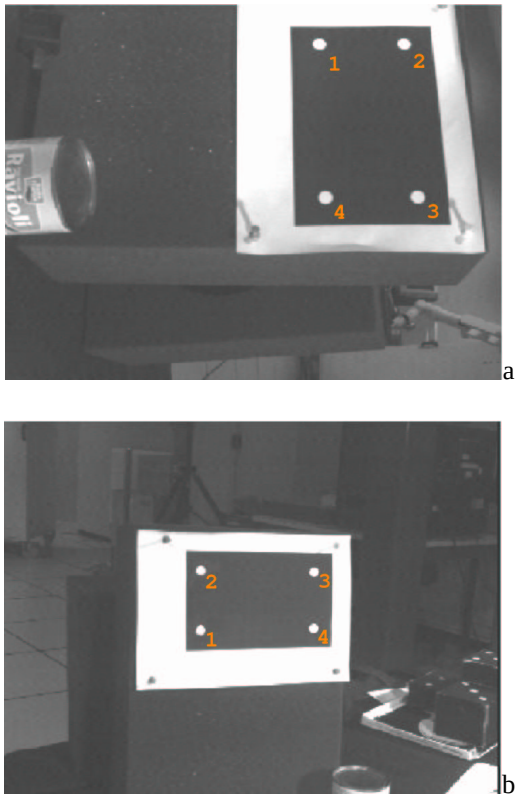

FIG. 4 – (a) Image intiale (b) Image désirée

La dernière exécution utilise la méthode complète (cf. Fig. 7). Lors de l'exécution de la dernière tâche, le contrôleur de pile anticipe la collision avec la butée. Il choisit alors de retirer la tâche $\mathbf{e_z}$, qui empêche le robot d'accomplir le mouvement selon l'axe optique qui lui permettrait d'éviter sa butée. On voit sur la figure 8 que la valeur optimale du critère correspond bien à $\mathbf{e_Z}$. La tâche $\mathbf{e_R}$ peut alors être menée à bien, en évitant facilement l'obstacle. La tâche $\mathbf{e_z}$ est ensuite remise dans la pile, et le robot rejoint sa position désirée.

6 Conclusion

Nous avons proposé dans cet article une nouvelle méthode pour éviter les obstacles au sens générique du terme, en particulier les butées articulaires. Lorsque le robot est loin de la position finale, il est commandé par une loi ne contraignant pas tous ses ddl, et les mouvements restants peuvent être facilement utilisés pour éviter les configurations indésirables. Lorsque le nombre de ddl diminue, un contrôleur de pile surveille l'exécution et peut le cas échéant retirer une des contraintes de la pile, pour dégager les ddl nécessaires à l'évitement. Ce schéma général de commande a ensuite été mis en œuvre sur un robot à six ddl avec une caméra embarquée. Les algorithmes

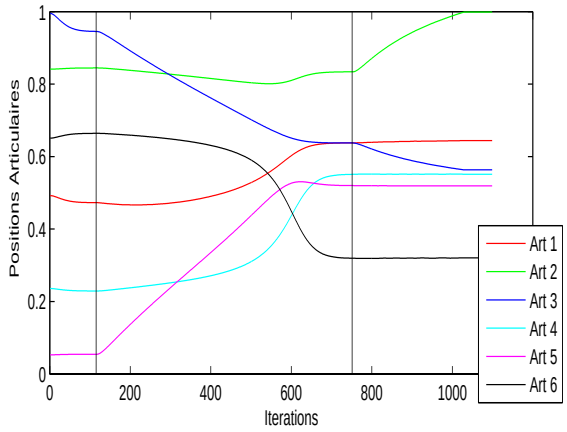

FIG. 5 – Trajectoires articulaires normalisées (les butées de chaque axe sont en 0 et 1) du robot lors d'une exécution utilisant la pile de tâches sans évitement d'obstacle. Le robot atteint sa butée lors de l'exécution de la troisième tâche, alors que deux ddl sont encore disponible (la tache $\mathbf{e_R}$ n'a pas encore été introduite dans la pile). Chaque trait vertical correspond à l'ajout d'une tâche dans la pile.

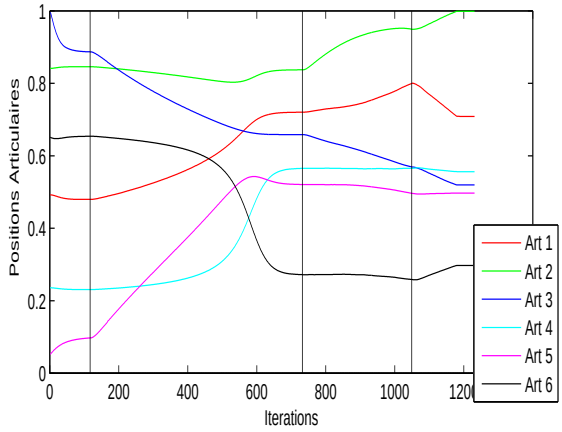

FIG. 6 – Trajectoires articulaires normalisées du robot lors d'une exécution utilisant la pile de tâches avec évitement d'obstacle simple. Le robot évite la butée tant que des ddl restent disponibles. La butée est atteinte lors de l'exécution de la dernière tâche, car il ne reste plus de ddl pour accomplir l'évitement. Chaque trait vertical correspond à l'ajout d'une tâche dans la pile.

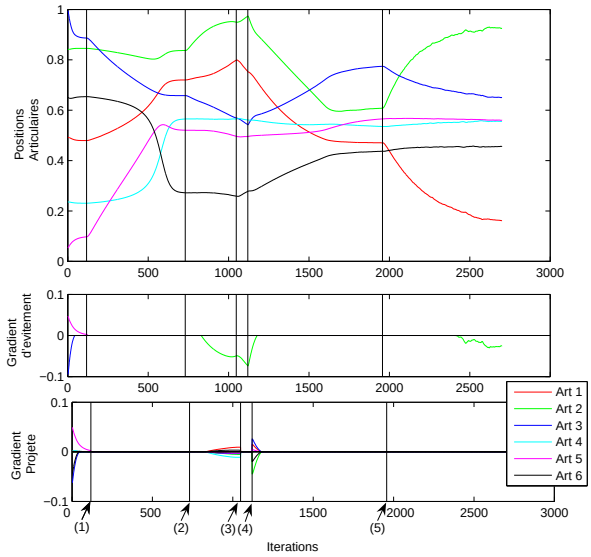

FIG. 7 – La position initiale est proche des butées du robot. L'évitement est donc activé dès la première tâche (avant l'événement (1)) pour replacer le robot dans l'espace acceptable. L'évitement est de nouveau activé lors de l'exécution de la troisième tâche (entre les événements (2) et (3)). On ajoute alors la dernière tâche (événement (3)). L'évitement est toujours actif (gradient non nul), mais, à cause du manque de ddl, le gradient projeté est nul. L'évitement n'est plus pris en compte. Le contrôleur détecte alors la collision (événement (4)), et retire la tâche $\mathbf{e_Z}$. Le robot rejoint alors la position désirée en remettant la tâche $\mathbf{e_Z}$ dans la pile (événement (5)), malgré la proximité de la butée.

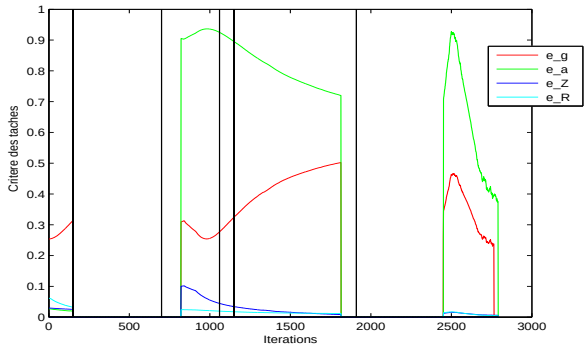

FIG. 8 – Evolution du critère calculé utilisé pour enlever une tâche de la pile lors de l'asservissement. Le critère présenté sur le graphe est la somme des deux critères (22) et (23). Au moment de retirer une tâche (itération 1230), le critère optimal correspond à la tâche $\mathbf{e_Z}$. Les critères ne peuvent etre calculé que lorsque la loi d'évitement n'est pas nul. Quand le gradient est nul, les citères valent $\mathbf{0}$.

classiques étaient incapables de terminer correctement la tâche demandée, alors qu'en utilisant la méthode proposée, l'exécution a pu se terminer correctement.
Les travaux que nous menons actuellement visent à valider cette approche pour un environement plus complexe, imposant d'éviter des obstacles plus nombreux et plus divers. Nous travaillons aussi à une automatisation du choix de la tâche à ajouter dans la pile, pour remplacer le choix *a priori* utilisé pour l'instant dans nos expériences. Le robot pourrait ainsi choisir la tâche convenant le mieux parmi un large choix de tâches possibles.

Références

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

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

[3] F. Chaumette. Image moments: a general and useful set of features for visual servoing. *IEEE Trans. on Robotics*, 20(4):713–723, August 2004.

[4] 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.

[5] 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. *IEEE Int. Conf. on Robotics and Automation (ICRA'03)*, 3:3929–3934, September 2003.

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

[7] 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.

[8] N. R. Gans and S. A. Hutchinson. An experimental study of hybrid switched approaches to visual servoing. *IEEE Int. Conf. on Robotics and Automation (ICRA'03)*, 3:3061–3068, September 2003.

[9] G. Hager. Human-machine cooperative manipulation with vision-based motion constraints. *Workshop on visual servoing, iros02*, October 2002.

[10] 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.

[11] 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.

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

[13] 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.

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

[15] B. Nelson and P.K. 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.

[16] L. Peterson, D. Austin, and D. Kragic. High-level control of a mobile manipulator for door opening. *IEEE Int. Conf. on Robotics and Automation (ICRA'03)*, 3:2333–2338, September 2003.

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

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

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

[20] B. Siciliano and J-J. Slotine. A general framework for managing multiple tasks in highly redundant robotic systems. In *ICAR'91*, pages 1211 – 1216, 1991.

[21] P. Soueres, V. Cadenat, and M. Djeddou. Dynamical sequence of multi-sensor based tasks for mobile robots navigation. *7th Symp. on Robot Control (SYROCO'03)*, 2:423– 428, September 2002.

[22] O. Tahri and F. Chaumette. Image moment: generic descriptors for decoupled image-based visual servo. *IEEE Int. Conf. on Robotics and Automation (ICRA'04)*, April 2004.