Cartes de visibilité pour la planification de stratégies de déplacement

Laurence Pelletier
INRIA, projet ICARE
06 902 Sophia Antipolis
Laurence.Pelletier@sophia.inria.fr

Résumé - *Dans cet article nous présentons un algorithme de visibilité que nous avons développé afin d'aider à planifier le déplacement d'un robot mobile en termes de tâches référencées capteur à enchaîner.*

Mots clé - stratégies de déplacement, commande référencée capteur, visibilité

1 Introduction

Compenser les incertitudes dans la navigation d'un robot mobile exige de tenir compte des capacités sensorielles du robot dès la phase de planification. Nous considérons ici le problème de la planification de stratégies de déplacement d'un robot mobile muni de capteurs extéroceptifs (caméra, capteurs de distance) dans un environnement structuré connu. Notre but est d'extraire automatiquement des informations du modèle de l'environnement pour décomposer le déplacement du robot en une séquence de tâches référencées capteur élémentaires à enchaîner. Pour cela nous fournissons un algorithme efficace pour calculer les régions depuis lesquelles les différents amers de l'environnement sont visibles, puis nous déterminons les tâches réalisables à partir des amers de chaque région pour calculer une séquence de tâches permettant de naviguer d'un point à un autre.

2 Motivations

La plupart des méthodes existantes de planification de mouvements produisent une trajectoire géométrique à partir d'une représentation supposée complète et exacte de l'environnement [4]. Or dans les situations réelles, de nombreuses erreurs liées au modèle de l'environnement, à la commande, ainsi qu'aux données sensorielles, interviennent. D'où l'intérêt de considérer en même temps planification de mouvements et commandes référencées capteur (qui permettent de générer des mouvements robustes vis-à-vis des différentes sources d'incertitude), pour déterminer des stratégies de déplacement.

Dans ce domaine de recherche récent qui rapproche commande, perception et planification, la seule approche multicapteur est l'approche probabiliste de Page [5], mais l'aspect commande y est tout juste évoqué rapidement. D'un autre côté, Bouilly ou Collin [1, 2] pour des ultrasons, et Kosecka [3] pour la vision proposent des approches de navigation à base de commandes référencées capteur, mais ces approches sont très fortement dépendantes des modélisations des capteurs utilisés, et sont peu facilement extensibles pour d'autres types de capteurs. C'est pourquoi dans le but de traiter de manière uniforme différents types de capteurs nous proposons grâce à un formalisme de commande générique, les *Liaisons Virtuelles* [8], d'exprimer le problème de planification de stratégies de déplacement en termes de planification de tâches référencées capteur.

3 Tâches référencées capteur

On peut réaliser la plupart des applications robotiques en environnement structuré au moyen d'une séquence de tâches référencées capteur. Nous avons développé une bibliothèque de tâches élémentaires [6] permettant de réaliser une trajectoire en lui associant une loi de commande référencée capteur robuste. Ces tâches sont définies par un type de capteur et un amer (motif composé de points ou segments de l'environnement), par exemple:

- tâches avec un télémètre: suivre un mur, tourner autour d'un coin;
- tâches visuelles: suivre une ligne, tourner autour d'une arête verticale, se positionner par rapport à une cible.

Pour un capteur géométrique C (caméra ou capteur de distance par exemple), fixé de manière rigide au robot, et retournant un signal $s = (s_1, ..s_k)$ dépendant uniquement de la position du robot $\overline{r}$ par rapport à son environnement local T, on peut écrire la différentielle des composantes scalaires de s comme un produit de torseurs:

$$\dot{s_i} = \frac{\partial s_i}{\partial \overline{r}} \dot{\overline{r}} = (H_{i,j}(C), u) \bullet (V_C, \omega_{F_C,F_T})$$

où $T_{ST} = (V_C, \omega_{F_C,F_T})$ est le torseur cinématique et $(H_{i,j}(C), u)$, appelé *Torseur d'Interaction*, représente l'interaction entre le capteur et l'environnement local du robot.

Grâce à la forme matricielle de cette équation, $\dot{s_i} = L_i^T.T_{ST}$, on peut synthétiser une loi de commande robuste et déterminer quels sont les mouvements libres, c'est-à-dire les mouvements permettant de garder le signal capteur désiré s_d constant. On définit ainsi de manière générique une *Liaison Virtuelle* entre un capteur géométrique et l'environnement local du robot, ce qui va nous permettre de traiter uniformément des tâches réalisées à partir de différents types de capteurs dans un seul formalisme. Ces *Liaisons Virtuelles* permettent de déterminer une bibliothèque de tâches élémentaires canoniques.

Par exemple pour réaliser un suivi de mur par asservissement visuel avec une caméra embarquée sur le robot, on sait qu'on a besoin de percevoir deux droites parallèles, ici la droite d'intersection entre le sol et le mur, et une plinthe.

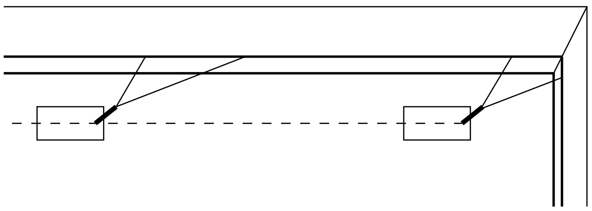

FIG. 1 – *Suivi de mur par asservissement visuel*

Liaison Virtuelle	Classe	Primitive 3D	Paramètres 3D
Prismatique	1 T	3 droites parallèles	$\begin{cases} a_{1i}, b_{1i}, c_{1i}, d_{1i} \\ a_{2i}, b_{2i}, c_{2i}, d_{2i} \end{cases}$ pour $i=1..3$
Motif 2D	Signaux capteur		Mouvements libres
3 droites	(ρ_i, θ_i) pour $i=1..3$		$\begin{pmatrix} a \\ b \\ c \\ 0 \\ 0 \\ 0 \end{pmatrix}$

Toutes ces tâches référencées capteur sont définies localement et doivent être enchaînées afin d'obtenir une stratégie globale de déplacement du robot. Pour cela, nous calculons off-line une décomposition exacte en cellules de l'environnement à partir du calcul des régions de visibilité des différents amers.

4 Visibilité d'un amer

Nous considérons un environnement polygonal borné composé d'obstacles polygonaux simples, convexes ou concaves, ayant deux à deux une intersection nulle. De nombreux algorithmes de calculs de visibilité existent [7], mais la plupart ne concernent que la visibilité de points et non de segments, ou alors ils ne traitent que des scènes d'objets en position générale, c'est-à-dire

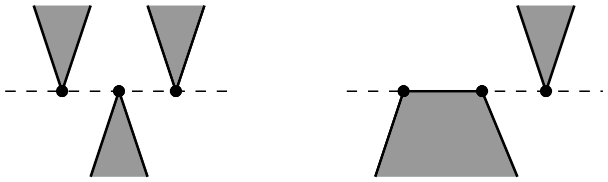

FIG. 2 – *Configurations dégénérées où des sommets de polygones sont alignés*

où les sommets des polygones ne sont pas alignés, ce qui exclut les cas présentés figure 2.

Ceci n'étant pas réaliste dans le cadre d'un environnement structuré comme celui dans lequel nous souhaitons faire évoluer le robot (cf l'exemple du modèle du laboratoire présenté figures 3 à 6),nous avons conçu un algorithme efficace pour calculer les polygones de visibilité des sommets et des arêtes de l'environnement en tenant compte de ces cas dégénérés.

Le polygone de visibilité d'un point P est l'ensemble des sommets de polygone de la scène vus depuis ce point et classés par ordre angulaire.

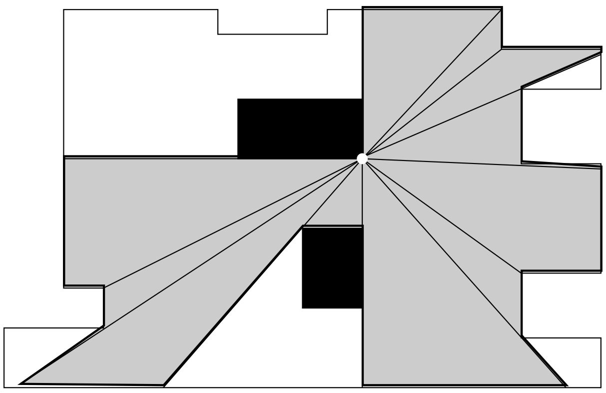

FIG. 3 – *Polygone de visibilité d'un sommet*

Soit H^+ le demi-plan défini par S ne contenant pas l'obstacle auquel appartient S. Le polygone de visibilité d'une arête S est l'ensemble des points M du demi-plan H^+ tels qu'il existe un rayon issu d'un point de S passant par M ne traversant aucun obstacle.

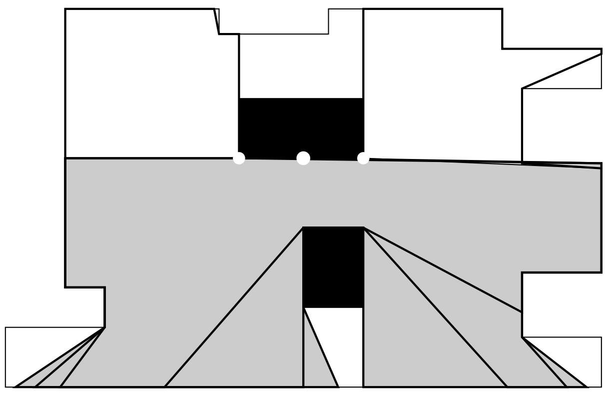

FIG. 4 – *Polygone de visibilité d'un segment*

Un algorithme naïf pour calculer les polygones de visibilité des sommets de l'environnement consiste en un balayage angulaire de rayons émis par chacun des sommets P pour déterminer les derniers points vus par P, puis en un réordonnancement de ces points afin d'obtenir un polygone simple.

Les rayons limites qui permettent d'obtenir les sommets du polygone de visibilité de P sont ceux qui passent par un autre sommet d'obstacle (figure 5).

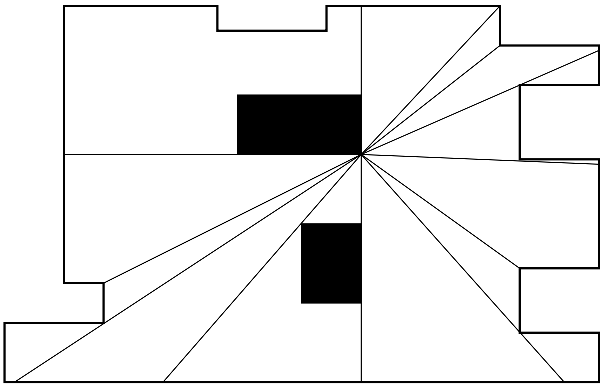

FIG. 5 – *Lancer de rayons pour un sommet*

De manière similaire, on peut déterminer les points critiques (représentés par des disques blancs figure 4) permettant de calculer le polygone de visibilité d'une arête S comme étant les points de S qui délimitent les zones d'ombre de S . Ce sont des points d'intersection d'une droite passant par deux sommets de l'environnement avec une arête de polygone (sur un obstacle ou sur le bord de la scène) (figure 6).

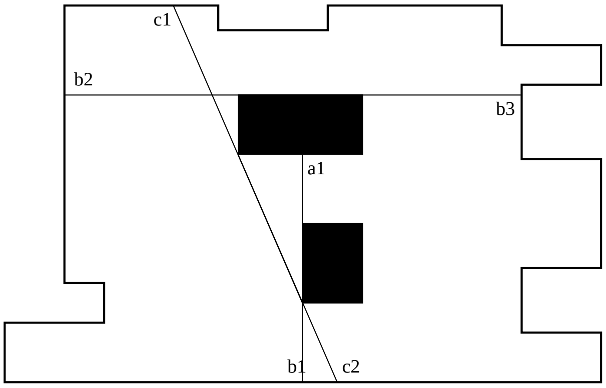

FIG. 6 – *Points critiques*

Pour éviter de recalculer les intersections des droites D passant par deux sommets de la scène avec tous les obstacles à chaque calcul de visibilité, nous avons défini un algorithme qui balaye toutes les paires de sommets (V_1, V_2) de l'environnement et stocke avec leur angle les points visibles dans la direction V_1->V_2 pour V_1 puis V_2->$V1$ pour V_2 grâce à un système de doubles dictionnaires. Dans `Dico` à chaque point correspond un dictionnaire de listes de points vus, dans lequel ces listes sont indexées par leur angle de vision. On récupère de la sorte aisément une liste complète des points vus par chaque sommet P et il reste à l'ordonnancer correctement pour construire le polygone de visibilité de P.

Durant ce balayage nous déterminons lors des calculs d'intersection les points critiques sur les arêtes de l'environnement pour calculer à faible coût les régions depuis lesquelles chacune des arêtes est visible.

```
Pour chaque sommet V1 de l'envt
  Dico(V1) = empty
  I_i = arete entrante de V_i
  O_i = arete sortante de V_i
  thetaMax = angle(I1,O1)

Pour chaque couple {V1,V2} de sommets
  creer la droite D(V1,V2)
  theta_i = angle(I_i,D)
  Si (theta1 <= thetaMax)
    L = intersection(D,polygones de l'envt)
    Dico(V1)(theta1) = PtsVus(V1,V2,L)
    Dico(V2)(theta2) = PtsVus(V2,V1,L)
```

où PtsVus(V1,V2, L) est la fonction retournant la liste V de points vus par V_1 dans le sens V_1 ->V_2:

```
Ordonner L
V = empty
Pred = V1
Parcourir L dans le sens V1 -> V2
  Tant que [Pred,Courant] ne coupe pas d'obstacle
    ajouter Courant a V
    # traitement cas degeneres cf fig 2
    Si (Pred sommet de P1) ET (Courant sommet de P2)
      Si (P1 et P2 pas dans le meme demi-plan
          par rapport a [V1V2])
        break
    Sinon Pred = Courant
```

Maintenant qu'on a pour chaque point un dictionnaire classant par angle les listes des points vus depuis P, il reste à les ordonner pour obtenir le polygone de visibilité. En effet à la sortie de l'algorithme les points sur chacune des listes du dictionnaire de P sont ordonnés depuis P vers les points visibles et cela ne nous permet pas toujours d'obtenir un polygone simple. Par exemple, figure 7 les flèches noires indiquent l'ordre souhaité, alors que dans le cas de plusieurs points alignés on obtient l'ordre indiqué par la flèche blanche.

Pour rétablir le bon ordonnancement des points il suffit de remarquer que dans un polygone de visibilité il ne peut pas y avoir deux arêtes virtuelles (c'est-à-dire deux arêtes n'appartenant à aucun obstacle) consécutives. Il doit toujours y avoir une arête physique (délimitant une portion d'obstacle) entre deux arêtes virtuelles.

Si l'on obtient deux arêtes virtuelles consécutives en concaténant dans l'ordre des angles croissants les listes de points vus du dictionnaire de P c'est que la liste correspondant à l'introduction de la deuxième arête virtuelle doit être inversée avant la concaténation (cf figure 7).

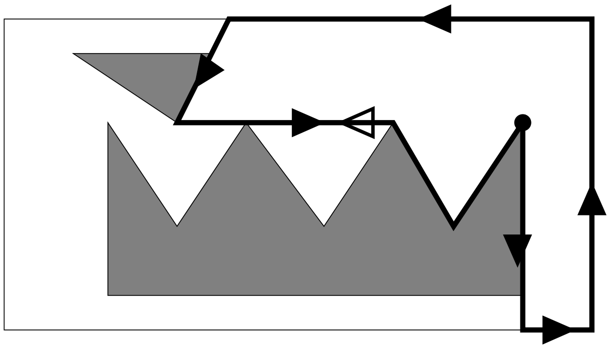

Fig. 7 – *Problème d'ordonnancement des points*

Lors de l'algorithme les points critiques (correspondants aux points d'intersection des droites D avec les arêtes de l'environnement) sont stockés sur les arêtes auxquelles ils appartiennent. Ainsi lorsque nous voulons calculer le polygone de visibilité d'une arête il suffit juste de calculer la réunion des polygones de visibilité de ses sommets et de ses points critiques.

5 Générer une séquence de tâches référencées capteurs

A partir de ces calculs de visibilité nous définissons une décomposition exacte en cellules de notre environnement [4]. Nous allons maintenant extraire un canal de navigation le long duquel nous pourrons déterminer les tâches référencées capteur réalisables en fonction des capteurs disponibles sur le robot.

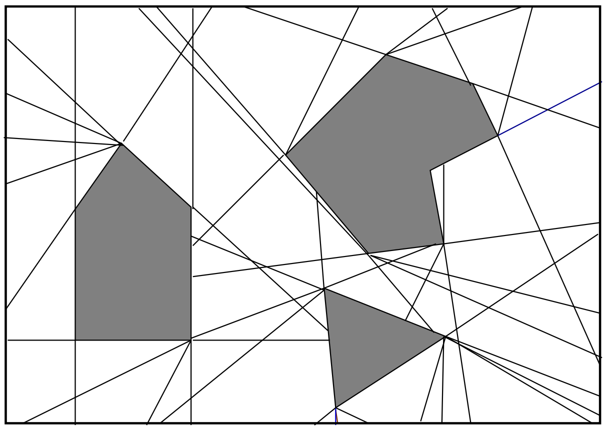

Fig. 8 – *Partitionnement de l'environnement*

En intersectant toutes les régions de visibilité des amers de l'environnement nous établissons un graphe de cellules. Chaque noeud contient une cellule (région homogène au sens des amers perçus) avec la liste des tâches référencées capteur élémentaires qui peuvent être réalisés avec les amers visibles dans cette cellule (primitives en gras figure 9).

Nous avons développé une interface graphique qui permet à l'utilisateur de définir un environnement et d'obtenir une carte de visibilité, en `C++` avec la bibliothèque de primitives de géométrie algorithmique LEDA [1]. On peut aussi utiliser l'algorithme de visibilité pour localiser le robot. Des exemples d'exécution du logiciel, et notamment du traitement des cas dégénérés, sont visibles à l'adresse `http://www.inria.fr/icare/personnel/pelletier/screenshots.html`.

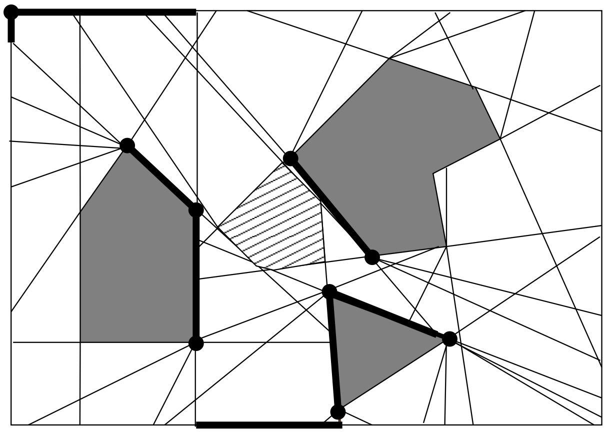

Fig. 9 – *Amers visibles depuis une région*

L'interface permettant de déterminer les tâches réalisables correspondant à un déplacement, grâce à la bibliothèque de tâches canoniques et à l'information donnée par l'utilisateur à propos des capteurs disponibles sur le robot, est en cours d'implémentation. L'utilisateur pourra alors fixer les paramètres des capteurs(portée, angle de vue, emplacement sur le robot...) pour calculer les lois de commande à appliquer au robot et la trajectoire désirée.

Il y a trop de paramètres pour qu'il soit envisageable de calculer de manière automatique les valeurs des signaux à réguler pour chacune des tâches référencées capteurs de la séquence, mais il est possible d'envisager de générer des corps de Tâches-Robots, cf [2], à compléter ensuite par l'utilisateur. Ceci devrait permettre d'aider à la gestion des événements discrets réglant le changement d'une tâche à une autre.

6 Conclusion

L'avantage principal de notre approche est que le prétraitement de visibilité ne dépend ni du nombre ni du type de capteurs utilisés, ni même des paramètres des capteurs (portée, angle de vue, emplacement sur le robot...). Le calcul des tâches réalisables est fait seulement dans un deuxième temps à partir des informations sur les capteurs que fournit l'utilisateur.

Cet outil peut également être facilement étendu à d'autres tâches référencées capteur sans nécessiter de nouvelle modélisation de l'environnement (aucun autre prétraitement) et permet de manipuler uniformément ces tâches du point de vue de la commande.

1. Library of Efficient Data types and Algorithms, `http://www.mpi-sb.mpg.de/LEDA/index.html`

Les zones de l'environnement où aucune tâche référencée capteur (amers hors de portee...) n'est réalisable pourront également être détectées.

Références

[1] B. Bouilly, T. Siméon, and R. Alami. A Numerical Technique for Planning Motion Strategies of a Mobile Robot in Presence of Uncertainty. In *Proc. IEEE ICRA*, Nagoya, Japan, May 1995.

[2] Ivan Collin. *Planification de Tâches-Robots pour Robots Mobiles en Environnement Structuré*. PhD thesis, Université de Technologie de Compiègne, Janvier 1995.

[3] Jana Kosecka. *A Framework for Modelling and Verifying Visually Guided Agents: Design, Analysis and Experiments*. PhD thesis, GRASP Laboratory, U. of Pennsylvania, March 1996.

[4] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1990.

[5] Lance A. Page and Arthur C. Sanderson. Robot Motion Planning for Sensor-Based Control with Uncertainties. In *Proc. IEEE ICRA*, Nagoya,Japan, May 1995.

[6] P. Rives, R. Pissard-Gibollet, and L. Pelletier. Sensor Based Tasks: From the Specification to the Control Aspects. In *6th Int. Symposium on Robotics and Manufacturing*, Montpellier, France, May 1996.

[7] Stéphane Rivière. *Calculs de Visibilité dans un Environnement Polygonal 2D*. PhD thesis, Université Joseph Fourier, Grenoble, Janvier 1997.

[8] C. Samson, B. Espiau, and Le Borgne M. *Robot Control: The Task Function Approach*. Oxford University Press, 1990.