Compte rendu technique
Evitement d'obstacles et suivi de trajectoires

Christophe Braillon

26 septembre 2003

Table des matières

Première partie

Introduction

Le CyCab avant les modifications

Avant la réalisation de ce travail, beaucoup d'applications destinées aux CyCab avaient été développées. Notamment un programme de planification et de suivi de trajectoires ([HPSL03], [HPS+03] et [Her03]). Pour pouvoir utiliser le planificateur, il fallait dans un premier temps faire une reconnaissance de l'environnement dans lequel le robot devait évoluer, ceci grâce à un télémètre à balayage. Ensuite le planificateur calculait en conséquence la trajectoire la plus adaptée pour atteindre le point d'arrivée.

Les principaux problèmes liés à l'exploitation du CyCab étaient d'un côté le manque de fiabilité de certains programmes, non pas au niveau algorithmique mais au niveau réalisation (il arrivait fréquemment qu'il y ait des "segmentation fault" durant l'exécution) ; d'un autre coté un certain manque d'ergonomie lié à l'utilisation de l'interface Homme/Machine développée pour effectuer les démonstrations. D'autre part la loi de commande permettant de suivre précisément la trajectoire précalculée ne prévoyait pas la présence d'obstacles autres que ceux rencontrés lors de la planification (piétons, voitures sur le parking,...). D'où la nécessité du travail dont fait l'objet ce rapport.

But du travail

Le but de ce travail est donc dans un premier temps de fiabiliser les applications (et d'améliorer leur ergonomie) destinées au CyCab (notamment le planificateur de trajectoires), puis de pouvoir intégrer au suivi de trajectoire un évitement ([KPBM03]) d'obstacles afin d'améliorer la sécurité, surtout vis-à-vis des piétons. Le robot devra être capable d'éviter un obstacle imprévu et de se recaler sur la trajectoire qu'il avait planifiée au départ. Ce document présentera donc deux parties distinctes correspondant aux deux parties du stage. La première partie traitera des problèmes présents dans les applications et les solutions apportées et dans la deuxième nous aborderons le problème de l'évitement d'obstacle et de son implémentation sur le CyCab.

Deuxième partie

Fiabilisation des applications et amélioration de l'interface

Chapitre 1

Fiabilisation des applications

1.1 Diagnostic des problèmes

Un des principaux problèmes sur les applications venait du planificateur. En effet, il arrivait qu'au début de la planification, il y ait un "segmentation fault". Le code du planificateur est un assemblage de morceaux de code provenant de divers endroits, de différentes époques (notamment la segmentation de cartes par diagramme de Voronoï, qui a été développé en 1990 au LAAS de Toulouse en C K&R, qui a du être adapté au GNU C...). De plus ce code très peu commenté est assez obscur et les structures de données utilisées ne facilitent pas le debuggage et sont propices aux erreurs. Après quelques journée de tests acharnés, il s'est avéré que le problème se situait au niveau du calcul du Voronoï.

Un autre problème, celui-ci plus gênant était que le robot avait une avance dans certains cas de plus d'un mètre sur sa trajectoire. Il arrivait aussi que le planificateur considère la configuration de départ ou celle d'arrivée en collision avec un obstacle alors qu'elles ne l'étaient pas. La planification étant faite à partir du centre des deux roues avant, et l'exécution de la trajectoire étant basée sur le centre des deux roues arrières, le problème semblait venir d'un mauvais changement de repère, même si une attention particulière aux problèmes d'interfacage entre les deux modules avait été portée.

Un dernier problème venait du fait que parfois dans le simulateur (et parait-il sur le robot réel), les calculs d'inversion de matrices (dans le filtre de Kalman de la position) renvoyait la valeur "NaN". Le calcul étant effectué par une fonction de la GSL (GNU Scientific Library), il est très difficile d'y remédier.

1.2 Solutions apportées

Le premier problème a été relativement vite résolu (trois jours). Il s'agissait d'un variable qui n'etait pas tout le temps initialisée dans le module de calcul du diagramme de Voronoï. Ce problème était peut être dû à l'adaptation de code C K&R au GNU C. L'initialisation de cette variable était faite dans la boucle interne de deux boucles imbriquées, ce qui posait parfois problème dans la boucle externe. Après déplacement de cette ligne d'initialisation, il s'est avéré que le planificateur fonctionnait parfaitement et ce dans tous les cas de figure.

Le deuxième problème, n'était pas au départ très visible. En effet, les trajectoires planifiées s'exécutaient a priori correctement. Seuls certains cas paraissaient problématiques sans pour autant que l'hypothèse de l'oubli de changement de repère ne soit évoquée, d'autant plus que la correspondance écran/réalité n'est pas aisée et ne permettait pas de pouvoir repérer un distance d'un mètre. Une recherche approfondie (de plusieurs semaines), et la liste des types d'entrées et de sorties de chaque module de la chaîne de calculs, ont permis de déterminer le changement de repère manquant et de le corriger. Après correction de cette erreur, le robot n'a plus refusé d'effectuer les trajectoires demandées et les a effectuées sans écart significatif. Il faudra tout de même régler la loi de commande permettant de suivre la trajectoire car les gains étaient réglés pour suivre une trajectoire avec un retard de plus d'un mètre et il s'avère que sur certaines manoeuvres la loi de commande ne puisse pas corriger la trajectoire (par exemple sur les demi-tours).

Le dernier problème, venant de la GSL n'a toujours pas été résolu. L'installation de la dernière version de la GSL a tout de même permis de diminuer considérablement la fréquence d'erreurs, sans pour autant l'avoir éliminé définitivement. Depuis cette installation, le problème ne s'est jamais produit sur le CyCab... Il faudra peut-être aussi vérifier que le compilateur ne soit pas fautif (avec certaines versions de GCC, il arrive que la bibliothèque se compile mal).

Chapitre 2

Amélioration de l'interface Homme/Machine

2.1 L'interface avant modifications

L'interface graphique avant les modifications apportées avait quelques inconvénients au niveau ergonomie. Le principal problème se situait au niveau du Joystick. A chaque manipulation il fallait recalibrer, même s'il venait de l'être quelques minutes auparavant... D'autre part l'interface de l'application de conduite manuelle n'offrait pas la meilleur sécurité possible. Pour freiner, il fallait appuyer sur deux touches consécutivement (flèche bas et entrée). Enfin l'application de suivi de trajectoire n'offrait pas toutes les options possibles pour les démonstrations (mise à jour de la carte pendant le suivi de trajectoire, évitement d'obstacles, démo prédéfinie).

2.2 Modifications de l'interface

La première modification apportée à l'interface homme/machine a été d'ajouter un système de sauvegarde de la calibration du joystick afin de ne pas avoir à refaire toutes les manipulations à chaque début d'application. La configuration est sauvegardée sous le chemin `/root/.joystick`. Dans ce fichier sont stockés les informations relatives au zéro du joystick, au maximum et au minimum des valeurs ainsi que la date de calibration. Cette date sert à invalider le fichier s'il est agé de plus d'une heure. S'il l'est, il est supprimé et les applications utilisant le joystick demandent une nouvelle calibration.

La deuxième modification concerne l'interface de conduite manuelle. La seule modification effectuée a consisté à faire en sorte que le bouton "Start

driving" devienne "Put breaks" lorsqu'on est en mode conduite et vice-versa.

La dernière modification porte sur le programme LocNPlanif (suivi de trajectoire avec ou sans évitement d'obstacle). L'écran principal a été modifié de manière à afficher toutes les indications concernant la configuration du logiciel (type de demo, évitement d'obstacles, mise à jour de la carte,...), et un menu permettant de paramétrer la démo a été ajouté. Les deux captures d'écran ci-dessous présentent ces modifications.

La partie modification de l'interface n'a pas posé de problème particulier mais a permis de gagner un temps considérable lors des manipulations. Le temps passé à rendre l'interface un peu plus ergonomique sera de toute manière largement rentabilisé (certaines manipulations ont vu leur temps d'exécution divisé par deux voir trois).

FIG. 2.1 – Ecran principal de LocNPlanif

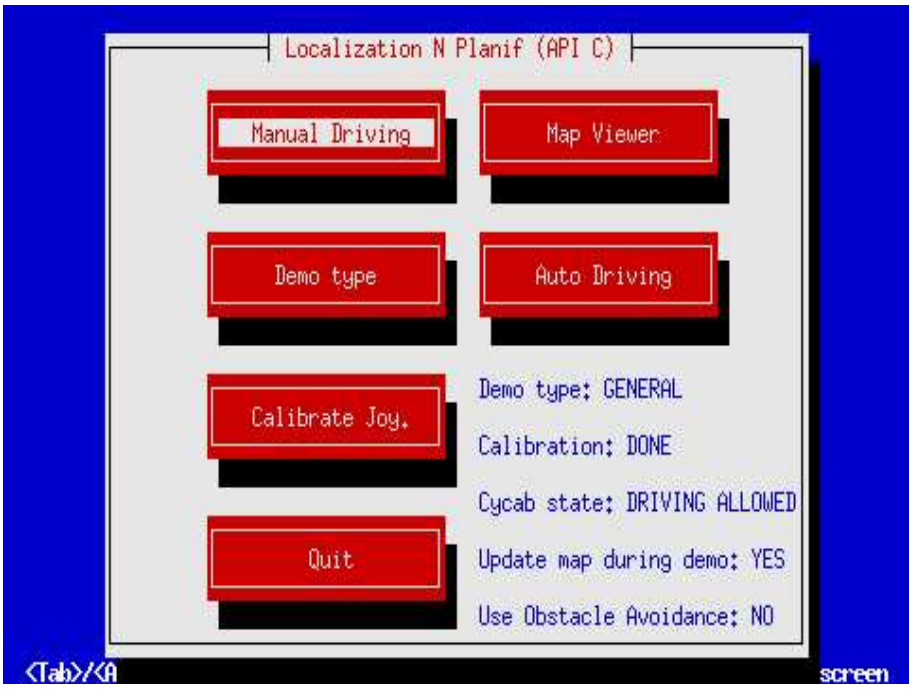
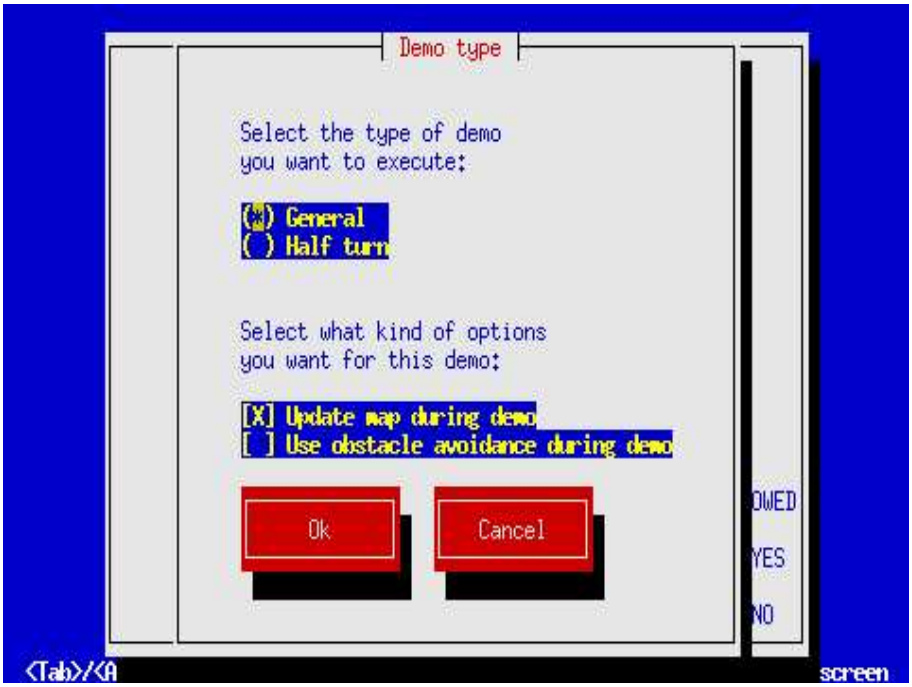


FIG. 2.2 – Sélection du type de démo dans LocNPlanif



Troisième partie

Evitement d'obstacles et suivi de trajectoires

Chapitre 3

L'évitement d'obstacles

3.1 Positionnement du problème

Pour intégrer l'évitement d'obstacles au suivi de trajectoire, il nous faut tout d'abord réaliser un module qui, à partir d'une commande demandée, nous donne une commande proche de celle voulue mais permettant tout de même d'éviter les obstacles éventuels. Il suffira de donner en entrée la commande calculée par le module de suivi de trajectoire et ensuite d'appliquer la commande sortie par le module d'évitement. La loi de commande actuelle du CyCab (commande du point plat) étant très sensible aux grandes erreurs (maximum 1m d'erreur pour la position et environ 15° pour l'angle), nous devons trouver une parade pour éviter que la trajectoire ne diverge completement si un obstacle vient perturber de manière prolongée le mouvement. Pour cela, nous disposons d'un suivi bayésien de trajectoire, beaucoup moins précis que la première loi mais moins sensible aux grandes erreurs. Il nous faudra donc sélectionner ces lois de commande selon l'amplitude de l'erreur. Le programme final de suivi de trajectoire comportera donc quatre modules :

- La loi de commande du point plat (flat control law)
- La loi de commande bayésienne
- Le module d'évitement d'obstacles
- Le multiplexeur

Les lois de commandes ne seront pas traitées dans ce document, car elle existaient avant d'intégrer l'évitement d'obstacles. Seuls l'évitement d'obstacles et le multiplexeur seront détaillés dans la suite.

3.2 Méthode utilisée

Nous adopterons pour le module d'évitement d'obstacles une approche probabiliste. En effet cette approche semble la plus apropriée à ce type de problèmes de part son aptitude à gérer les incertitudes et incomplétudes et de part la flexibilité de description qu'elle propose. Le module nous donnera, selon les différentes observations et la commande désirée, la probabilité qu'une commande soit la plus appropriée pour suivre la trajectoire tout en évitant les éventuels obstacles imprévus.

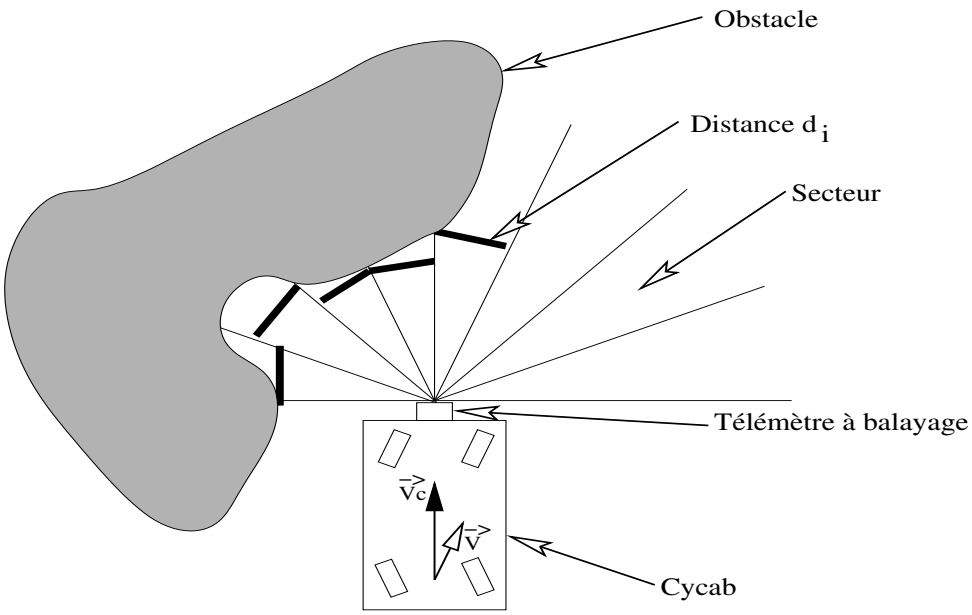
3.3 Définition du modèle

3.3.1 Notations

Le CyCab étant commandé en vitesse et angle de braquage, nous noterons la variable aléatoire représentant la commande envoyée au robot $\vec{V} = \begin{pmatrix} v \\ \phi \end{pmatrix}$ où v représente la vitesse linéraire du robot et ϕ son angle de braquage.

Nous utiliserons bien évidemment les données laser fournies par le télémètre à balayage (Sick), les données étant trop nombreuses, nous découperons le demi espace devant le robot en n secteurs. Nous aurons donc n variables aléatoires notées $D_k, k = 1 \ldots n$ représentant le minimum des distances mesurées sur le secteur.

Enfin nous noterons $\vec{V_c} = \begin{pmatrix} v_c \\ \phi_c \end{pmatrix}$ la variable aléatoire représentant la commande désirée où v_c et ϕ_c représentent respectivement la vitesse et l'angle de braquage désirés. Voci un exemple illustrant ces notations :



3.3.2 Modélisation

Nous cherchons en réalité la distribution de probabilité des commandes étant donné les n observations faites ainsi que la vitesse désirée. Nous cherchons donc $P(\vec{V} \mid D_1, \ldots, D_n, \vec{V_c})$. Cette distribution nous donnera (en recherchant son maximum), la commande la plus apropriée à envoyer au CyCab.

Nous considèrerons dans la suite de ce document que $D_1, \ldots, D_n$ et $\vec{V_c}$ suivent une loi uniforme, ce qui correspond à la distribution exprimant le moins de connaissances a priori.

Pour cela nous allons introduire une variable aléatoire hypothèse H prenant ses valeurs dans $\{1, \ldots, n\}$. H vaut k si et seulement si $\vec{V}$ n'est calculé qu'en fonction de D_i.

Nous obtenons donc :

$$\begin{aligned} P(\vec{V} \mid D_1, \ldots, D_n, \vec{V_c}) &= \sum_{i=1}^{n} P(\vec{V}, H = i \mid D_1, \ldots, D_n, \vec{V_c}) \\ &= \sum_{i=1}^{n} P(H = i \mid D_1, \ldots, D_n, \vec{V_c}) P(\vec{V} \mid D_1, \ldots, D_n, \vec{V_c}, H = i) \end{aligned}$$

On considère que l'évènement $H = i$ suit une loi uniforme afin que tous les D_k aient la même influence a priori sur le contrôle. Nous pouvons donc

simplifier l'expression ci-dessus en :

$$P(\vec{V} \mid D_1, \ldots, D_n, \vec{V}_c) = \sum_{i=1}^{n} P(H = i) P(\vec{V} \mid D_i, \vec{V}_c, H = i)$$

Avec l'hypothèse que $P(H = i)$ est constante, on obtient donc la formule :

$$\boxed{P(\vec{V} \mid D_1, \ldots, D_n, \vec{V}_c) \propto \sum_{i=1}^{n} P(\vec{V} \mid D_i, \vec{V}_c, H = i)}$$

3.3.3 Définition des lois de probabilités

Nous ferons l'hypothèse à partir de maintenant que $P(\vec{V} \mid D_i, \vec{V}_c, H = i)$ est une gaussienne centrée en $\vec{\mu}_i = \begin{pmatrix} \mu_{v_i} \\ \mu_{\phi_i} \end{pmatrix}$ et de matrice de covariance $\Sigma_i = \begin{pmatrix} \sigma^2_{vv_i} & \sigma^2_{v\phi_i} \\ \sigma^2_{\phi v_i} & \sigma^2_{\phi\phi_i} \end{pmatrix}$ les paramètres $\vec{\mu}_i$ et Σ_i de la gaussienne seront définis de manière à proposer une commande sûre, c'est à dire permettant d'éviter un obstacle.

Les espérances

Nous voulons que, lorsque la distance d_i est grande (i.e. il n'y a aucun danger car l'obstacle est éloigné), la commande envoyée au CyCab soit la plus proche possible de celle désirée. Par contre, si d_i est petite, le robot doit braquer de manière à éviter l'obstacle. Il apparaît donc naturel que $\vec{\mu}_i$ dépende de la distance d_i. A cela nous ajouterons trois autres paramètres : une distance m_i, une autre M_i, et une dernière S_i ces trois distances représentent respectivement la distance critique à partir de laquelle il y a un fort danger de collision, la distance à partir de laquelle on considère que l'objet vu n'est plus un danger et n'est donc plus pris en compte et enfin la distance de sécurité à partir de laquelle le robot doit s'arrêter car aucune manoœuvre ne permet d'éviter l'obstacle.

Dans un premier temps la sigmoïde :

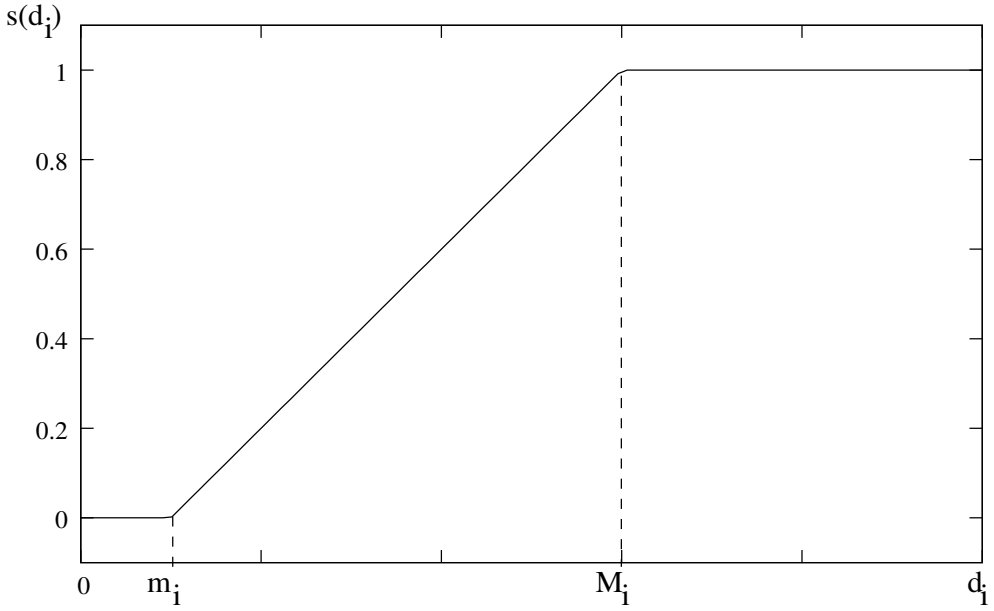
$$s(t) = \frac{1}{1 + e^{-\beta(t-\alpha)}}$$

paraissait la meilleure solution pour définir l'espérance de la loi, mais il s'est avéré qu'elle était "difficile à contrôler", en effet les réglages des paramètres

ne sont pas totalement intuitifs. Nous utiliserons donc une fonction linéaire par morceaux de la forme :

$$s_i(t) = \begin{cases} 0 & \text{si } t \leq m_i \\ \max(\min(\frac{\beta}{M_i - m_i}(t - m_i), 1), 0) & \text{sinon} \end{cases}$$

Voici l'allure des s_i :



On utilisera donc comme composantes de l'espérance :

$$\mu_{v_i} = v_c s_i(d_i)$$

$$\mu_{\phi_i} = \begin{cases} (\phi_c + \phi_{max}) s_i(d_i) - \phi_{max} & \text{si } i \leq \frac{n}{2} \\ (\phi_c - \phi_{max}) s_i(d_i) + \phi_{max} & \text{sinon} \end{cases}$$

Où ϕ_{max} est l'angle de braquage maximal du CyCab.

Les covariances

La matrice de covariance de chaque loi indique la prépondérance des $P(\vec{V} \mid D_i, \vec{V_c}, H = i)$. Plus la covariance est "grande", moins la loi aura d'influence sur le résultat final. Et à l'inverse plus la covariance est "petite", plus la loi aura d'influence. Pour un secteur donné, nous voulons que plus l'obstacle est proche, plus la commande donnée par la loi de probabilité associée au secteur soit prépondérante. D'autre part, nous considerons que

les deux composantes de $\vec{V}$ ne sont pas dépendantes, nous travaillerons donc avec une matrice de covariance diagonale de la forme :

$$\Sigma_i = ((1-\epsilon)s_i(d_i) + \epsilon) \begin{pmatrix} \sigma^2_{v,max} & 0 \\ 0 & \sigma^2_{\phi,max} \end{pmatrix}$$

3.4 Mise en pratique

Les paramètres des lois ont été estimées grâce à une interface graphique, permettant d'avoir instantanément l'allure de $P(\vec{V}, D_1, \ldots, D_n, \vec{V_c})$. Voici quelques exemples d'allures de lois de probabilité. Les paramètres choisis sont les suivants :

$$n = 9$$

$$\beta = 1.00$$

$$m_i = \sqrt{(1.0\cos(\tfrac{\pi(i+\frac{1}{2})}{n}))^2 + (1.5\cos(\tfrac{\pi(i+\frac{1}{2})}{n}))^2}$$

$$M_i = \sqrt{(4.0\cos(\tfrac{\pi(i+\frac{1}{2})}{n}))^2 + (6.0\cos(\tfrac{\pi(i+\frac{1}{2})}{n}))^2}$$

$$S_i = \sqrt{(0.6\cos(\tfrac{\pi(i+\frac{1}{2})}{n}))^2 + (0.2\cos(\tfrac{\pi(i+\frac{1}{2})}{n}))^2}$$

$$\phi_{max} = 0.38rad$$

$$v_{max} = 2.00m.s^{-1}$$

$$\sigma^2_{v,max} = 5.6$$

$$\sigma^2_{\phi,max} = 1.5$$

Une fois tous les paramètres des lois définis, l'implémentation de l'algorithme n'est plus un problème. Il suffit de calculer le vecteur $\vec{V}$ pour lequel la probabilité est maximale, c'est à dire qu'il faut tout simplement effecuter une recherche de maximum. Pour des raisons de simplicité et d'efficacité (peut être y a-t-il plus efficace), nous discrétiserons la loi de probabilité avec un tableau carré. La recherche de maximum devient alors un simple parcours (en $O(n)$, où n est la taille du tableau). L'algorithme utilisé est le suivant :

FIG. 3.1 – $v_c = 1.0$ $\phi_c = 0.2$ $D_1 = \ldots = D_n = \infty$

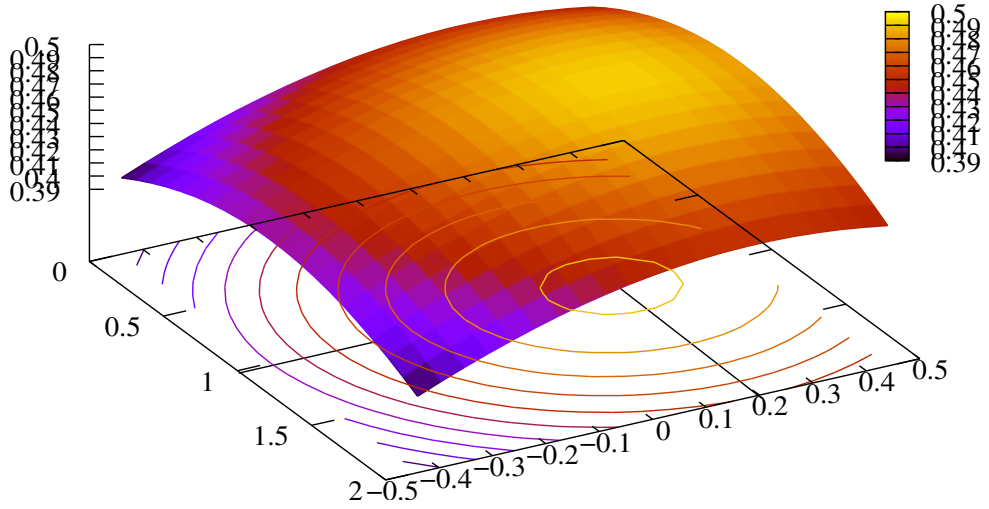


FIG. 3.2 – $v_c = 1.0$ $\phi_c = 0.0$ $D_1 = \ldots = D_3 = D_5 = \ldots = D_n = \infty$ $D_4 = 2.7$

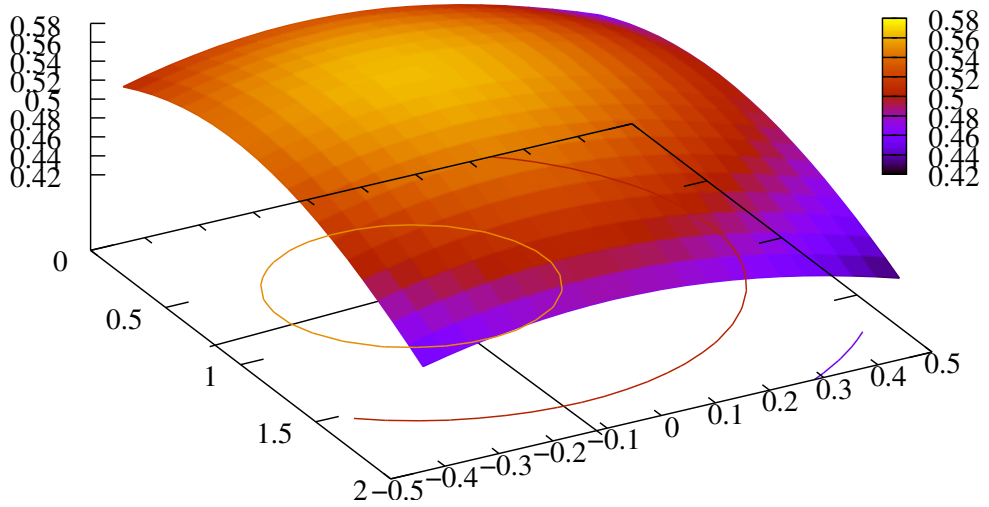


FIG. 3.3 – $v_c = 1.0$ $\phi_c = 0.0$ $D_1 = \ldots = D_2 = D_5 = \ldots = D_n = \infty$ $D_3 = D_4 = 1.2$

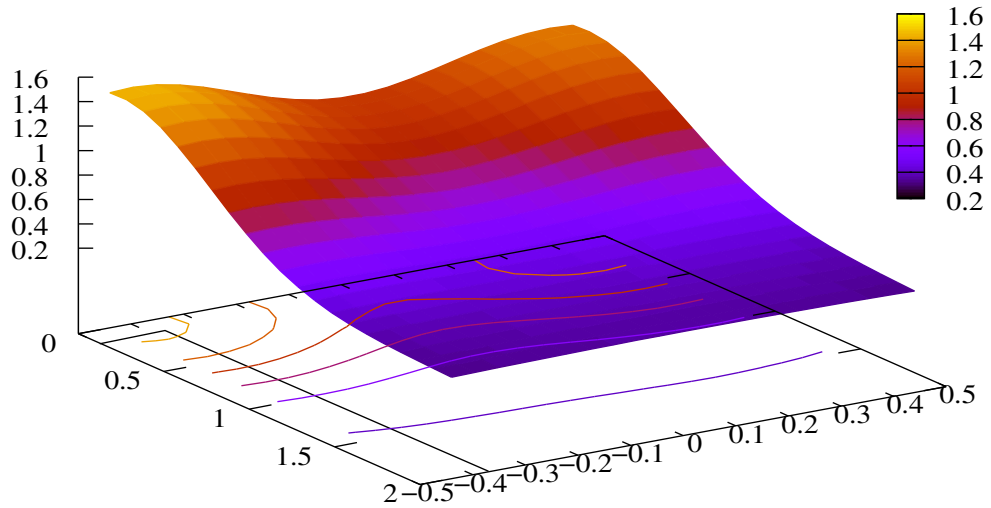
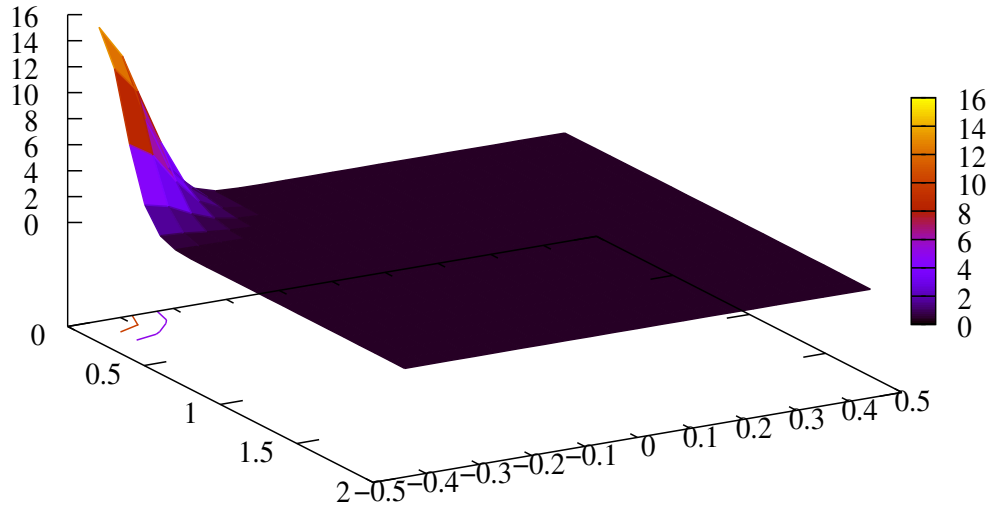


FIG. 3.4 – $v_c = 1.0$ $\phi_c = 0.0$ $D_1 = \ldots = D_3 = D_5 = \ldots = D_n = \infty$ $D_4 = 1.0$



```
Déclaration tableau t[TAILLE][TAILLE]

Pour chaque zone i faire
  Si D_i<=S_i alors
    v<-0
    fin
  fin Si
fin Pour

Pour i allant de 0 à TAILLE-1 faire
  Pour j allant de 0 à TAILLE-1 faire
    v<-V_max*i/(TAILLE-1)
    phi<-(2*PHI_max*j)/(TAILLE-1)-PHI_max

    t[i][j]<-Somme(P(V|Vc,Di,H=i),i=1..nbZones)
  fin Pour
fin Pour

max_V<-0
max_PHI<-0

Pour i allant de 0 à TAILLE-1 faire
  Pour j allant de 0 à TAILLE-1 faire
    Si t[i][j]>t[max_V][max_PHI] alors
     max_V<-i
     max_PHI<-j
    fin Si
  fin Pour
fin Pour

v<-V_max * max_V/(TAILLE-1)
phi<-(2*PHI_max*max_PHI)/(TAILLE-1)*PHI_max
```

Il s'est avéré que l'algorithme utilisé de cette manière répondait bien aux spécifications. L'évitement d'obstacles a dans un premier temps été testé sur le simulateur (voir figure ci-dessous). Les résultats étant satisfaisants en simulations, nous avons pu passer aux tests grandeur nature.

Le robot a été testé sur le parking de l'INRIA avec des obstacles statiques (voitures garées, cartons,...) et des obstacles dynamiques (cartons lancés de-

FIG. 3.5 – Evitement d'obstacles avec v_c constante et ϕ_c nul

vant le CyCab, piétons,...). La réactivité du système s'est révélée plus que satisfaisante, car le CyCab est capable d'éviter un piéton arrivant en courant et passant à quelques centimètres du véhicule (et ce même si le CyCab est à vitesse maximale). Cependant un problème (qui n'est pas lié a la méthode employée pour l'évitement d'obstacle) est apparu lors des essais. Le CyCab est équipé d'un télémètre à balayage à l'avant et il exite une zone morte dans laquelle le télémètre ne voit pas, Ceci est parfois problématique lorsque le télémètre a dépassé un obstacle et que le robot braque complètement du côté de l'obstacle, le robot risque de alors percuter l'obstacle par le côté. On peut remédier à ce problème facilement en équipant par exemple le CyCab d'autres télémètres voire de proximètres à ultrasons sur les cotés. Pour l'instant, nous avons limité la vitesse de braquage, ce qui permet d'éviter ces collisions.

Chapitre 4

Le multiplexage des correcteurs

4.1 Les correcteurs

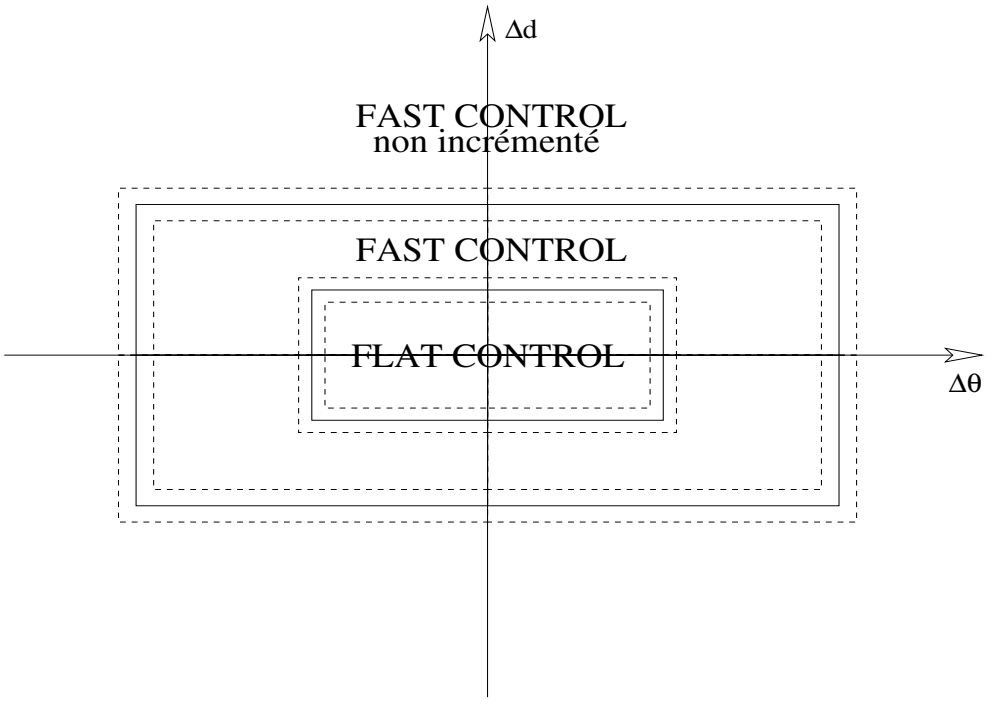
Deux types de suivi de trajectoire ont été implémentés pour le CyCab : une loi de commande controlant la derivée troisième du point plat (FlatControl) et une loi linéaire (FastControl). La première est une loi très précise donnant de très bons résultats, mais ne supportant pas les grosses erreurs (en particulier les erreurs en angle). On ne peut donc pas utiliser directement cette loi de commande en combinaison avec l'évitement d'obstacles. En effet, une déviation importante due à un obstacle statique a pour conséquence de faire diverger la loi de commande. La deuxième loi a une moins bonne précision mais est beaucoup plus tolérante que la première. Elle est capable de rattraper des erreurs de plusieurs mètres et de plusieurs radians sans problème. D'où l'idée de multiplexer ces deux lois afin d'obtenir de bons résultats quelle que soit l'erreur. Il faudra aussi ajouter une troisième loi identique à la deuxième mais où le point de référence reste bloqué. Cette troisième loi sera utilisée quand l'erreur sera trop grande afin que quand le robot est bloqué l'erreur soit bornée. On filtrera aussi l'accélération et la vitesse de braquage car quand un obstacle est évité, il ne faut pas que le robot ne change trop brutalement de vitesse ou d'angle de braquage, d'une part pour des questions de confort et d'autre part pour ne pas percuter l'obstacle avec l'arrière du CyCab.

4.2 Le multiplexeur

Le multiplexeur a pour but de donner la main à la loi la plus appropriée pour le suivi de trajectoire. En effet, les deux lois disponibles n'ont pas le même comportement aux erreurs. La première (Flat Control) est sensible aux

grosses erreurs, mais très précise. Le deuxième (Fast Control) est efficace en cas de grosses erreurs et moins précise. L'idée consiste donc à changer de loi à partir d'un certain seuil, tout en ajoutant une sorte d'hystérésis afin de la pas osciller entre les deux lois. Le critère de changement de loi se fait sur la distance entre la position réelle et la position de référence, et sur la différence d'angle entre l'angle courant et l'angle de référence. Ceci est résumé sur la figure ci-dessous :

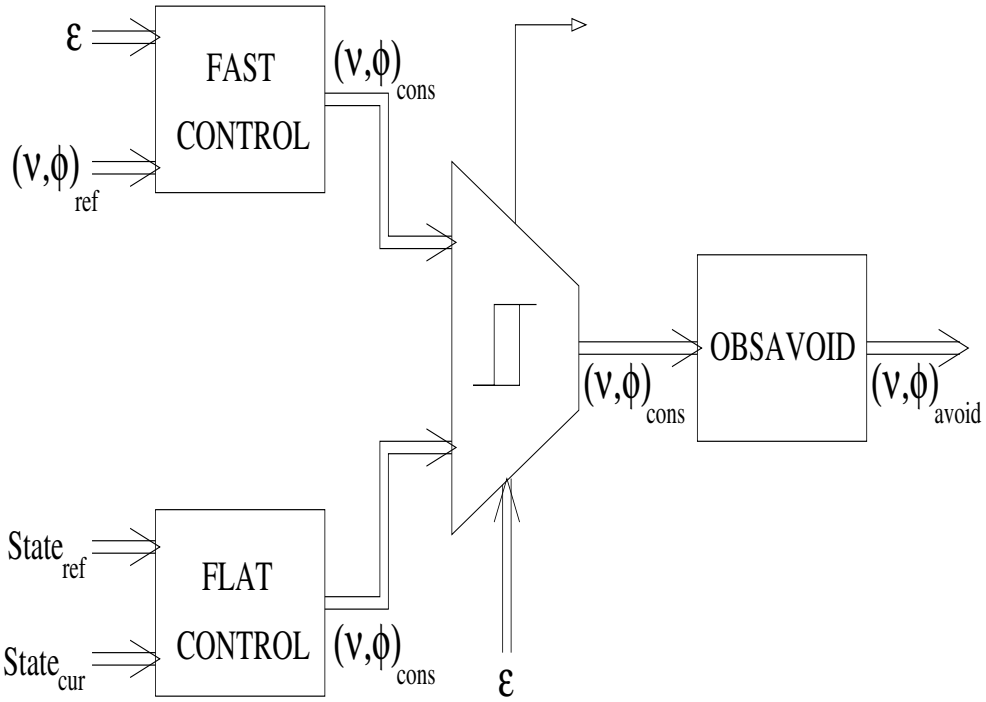
FIG. 4.1 – Principe de l'hystérésis



Le principe de fonctionnement du multiplexeur consiste à calculer l'erreur $\vec{\epsilon} = \begin{pmatrix} x_{reel} - x_{ref} \\ y_{reel} - y_{ref} \\ \theta_{reel} - \theta_{ref} \end{pmatrix}$ et en fonction de ces composantes calculer la correction à apporter à la trajectoire par l'un ou l'autre des deux correcteurs. Si l'erreur est trop grande, il faut aussi arrêter l'avancement du point de référence. Ce cas se produit notamment quand le robot reste bloqué devant un piéton pendant un certain temps ; si le point de référence continue d'avancer, lorsque le robot sera débloqué, il risque de ne jamais pouvoir rattraper le retard accumulé et donc de ne pas exécuter correctement la trajectoire planifiée.

En sortie du multiplexeur, on applique la méthode d'évitement d'obstacles afin que le robot puisse dévier de sa trajectoire initiale en cas d'obstacle imprévu. Ceci semble suffisant au premier abord mais il s'avère, que lorsque le robot est fortement ralenti pas l'évitement d'un obstacle, il accélère très fortement et braque vite pour rattraper le retard (une fois l'obstacle franchi). Il faut donc rajouter une limite à l'accélération et à la vitesse de braquage afin de la pas avoir de secousses trop violentes et que le robot ne se retourne (cette situation ne peut peut être pas se produire sur le CyCab, mais sur un autre type de véhicule, le problème pourrait très bien se produire). Le fonctionnement du multiplexeur est résumé sur la figure suivante. Il a fallu ajouter un filtre passe-bas pour limiter les variations d'angle de braquage, en effet la loi de commande et l'évitement d'obstacles étant en concurrence, l'angle de braquage oscille fortement et le filtre passe-bas permet d'éviter ces oscillations sans pour autant perturber l'exécution de la trajectoire.

FIG. 4.2 – Architecture du multiplexeur



Chapitre 5

Résultats expérimentaux

5.1 Les données expérimentales

5.2 Inconvénients de la méthode

5.3 Améliorations possibles

Bibliographie

[Her03] J. Hermosillo. *Planification et exécution de mouvements pour un robot bi-guidable : une approche basée sur la platitude différentielle*. PhD thesis, Grenoble (FR), June 2003.

[HPS+03] J. Hermosillo, C. Pradalier, S. Sekhavat, Ch. Laugier, and G. Baille. Towards motion autonomy of a bi-steerable car : Experimental issues from map-building to trajectory execution. 2003.

[HPSL03] J. Hermosillo, C. Pradalier, S. Sekhavat, and C. Laugier. Experimental issues from map building to trajectory execution for a bi-steerable car. 2003.

[KPBM03] C. Koike, C. Pradalier, P. Bessière, and E. Mazer. Proscriptive bayesian programming application for collision avoidance. 2003.