N° 04 MT 6002

Intégration des fonctions

« Anticollision » et « Gestion des inter-distances »

Laurent Trassoudaine
Roland Chapuis

Décembre 2005

A. K. AIJAZI
S. ALIZON
C. BLANC
R. CHAPUIS
F. CHAUSSE
O. DUVAL
J. LANEURIT
L. TRASSOUDAINE

Table des matières

1 Introduction

Le présent rapport est une synthèse des activités spécifiques du LASMEA sur le volet intégration de fonction. La problématique consiste en l'implémentation de briques de perception artificielle afin d'aboutir à des fonctions d'assistance à la conduite. Deux fonctions sont visées dans le cadre de cette réalisation : « Anticollision » et « Gestion des inter-distances ». Ces fonctions doivent être validées à bord du véhicule expérimental VELAC. Précompétitives, ces implémentations temps réel de perception multisensorielle doivent ouvrir la voie aux industriels pour le développement de nouveaux produits.

Ces recherches bénéficient des conclusions du thème 1, « perception », de l'action Arcos. Les briques scientifiques et techniques sélectionnées sur la base d'une évaluation des résultats obtenus sur des études de cas références sont implantées dans des démonstrateurs réalistes. Les briques logicielles du LASMEA portent sur des méthodes de fusion de données tant pour l'aspect localisation sur la chaussée (vision artificielle + capteurs proprioceptifs) que pour la détection et la localisation d'objets sur la chaussée (télémétrie laser + vision). Elles ont été implantées dans VELAC afin de conduire à la réalisation des deux fonctions d'assistance citées.

Le rapport est scindé en cinq chapitres : trois sont dédiés aux méthodes spécifiquement développées pour les fonctions d'assistance, deux aux aspects plus technologiques. Sur le premier volet, on retrouve la localisation du véhicule sur la chaussée, la détection d'obstacles et une méthode multisensorielle pour le suivi. Le second reprend les modifications spécifiques apportées à notre plate-forme de développement D-BITE ainsi que les actions mises en œuvres à bord du véhicule expérimental VELAC.

2 Localisation de véhicule sur la voie par reconnaissance visuel du marquage et combinaison de capteurs proprioceptifs

2.1 Introduction

Le LASMEA est impliqué dans des actions de reconnaissance de marquage tant à des fins civiles que militaires dans des contextes divers (routes marquées ou non) depuis le début des années 90 [1,3].
Les approches développées dans la littérature ont montré de bons résultats dans les situations « favorables » mais souffrent cependant de faiblesses dans les situations réalistes courantes (qualités des marquages, conditions d'éclairement). Bien souvent, les approches exploitent la géométrie normalisée de la chaussée afin de contraindre (et donc de rendre plus sures) les détections de ces marquages. Une des difficultés reste néanmoins le grand nombre de paramètres dynamiques définissant l'apparence de la chaussée dans l'image. En effet, aux caractéristiques propres à la route (courbure horizontale, voire verticale, etc), s'ajoutent les paramètres de localisation de la caméra (et donc du véhicule) qui l'observe. Les modèles complexes engendrés sont donc difficiles à stabiliser.

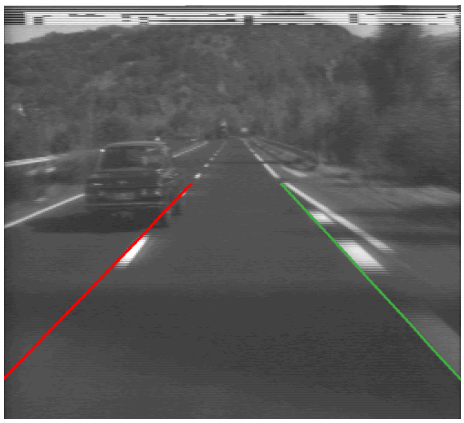


Figure 1 - Exemple de reconnaissance de la chaussée

Au LASMEA, nous avons développé une approche permettant de gérer de manière efficace un tel modèle, selon une approche statistique des bords de la chaussée dans l'image. Les stratégie de recherche permettent, de plus, de focaliser la recherche des bords dans des zones de l'image de telle sorte de l'on minimise les possibilités de fausses détections [4, 5, 6, 7]. La Figure 1 montre un exemple de sortie de l'algorithme.

2.2 Localisation du véhicule sur la voie

La reconnaissance des bords de la voie n'étant pas une fin en soit, il a été nécessaire, dans le cadre d'ARCOS, de développer une procédure de localisation du véhicule sur sa voie (dans le cadre de la fonction « prévention des sorties de route »). Celle-ci permet de fournir à chaque instant, à partir de la détection de la route dans l'image (voir Figure 2):

- la position réelle du véhicule par rapport aux deux bords de sa voie de circulation,
- l'angle de cap du véhicule,
- la courbure de la route à l'avant du véhicule,
- la largeur de cette voie de circulation,
- l'angle de tangage du véhicule.

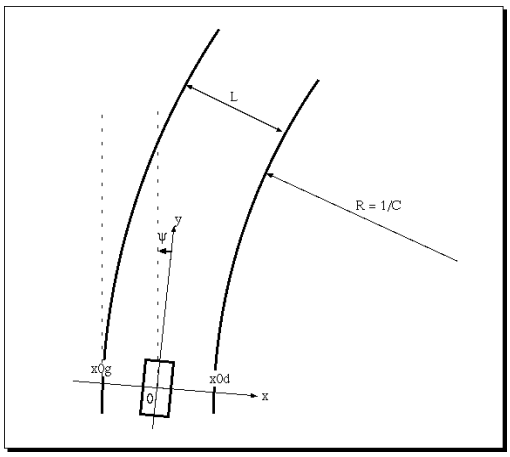


Figure 2 . Paramètres 3D utilisés

L'approche que nous avons développée [8], s'appuie sur le lien statistique existant entre les détections faites dans l'image, et les paramètres de localisation précédents (appelés 3D par souci de simplification). En effet, notre approche de reconnaissance de la route repose sur un modèle d'apparence statistique de la route dans l'image : un ensemble de configurations des paramètres 3D va donner un ensemble d'apparences de la route dans l'image (voir [2] pour les détails).
Une détection de la route dans l'image va donc correspondre à une configuration des paramètres 3D que l'on obtient par une simple observation de Kalman exploitant le lien statistique caractérisé par une matrice jacobienne (la fonction de passage étant non linéaire)

Cette approche originale est cependant difficile à valider de manière expérimentale en terme de précision. Grâce aux collaborations internes à ARCOS (notamment le LIVIC), nous avons eu l'opportunité de pouvoir caractériser la précision de la localisation fournie par notre algorithme grâce à un simulateur (voir Figure 3). Les paramètres du véhicule étant connus à chaque instant, la comparaison avec ceux fournis par l'algorithme est ainsi facilitée (une telle comparaison n'est en effet pas possible en situation réelle). Une séquence de 3000 images environ a été utilisée dans des configurations de courbure importantes (simulation du circuit de du Giat à Satory).

Figure 3 - exemple d'image du simulateur du LIVIC utilisé

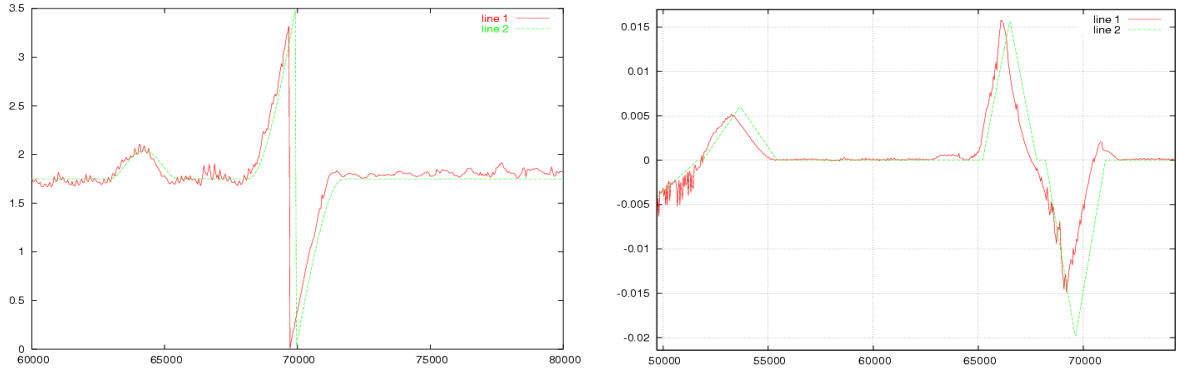


Figure 4 – (a) localisation latérale du véhicule, (b) : estimation de la courbure de la chaussée. En, vert : valeur réelle, en rouge : valeur estimée.

La Figure 4a montre un zoom de la position latérale réelle et estimée du véhicule par l'algorithme sur environ 500 images de la séquence. La forte variation est due à un changement de voie du véhicule.
La position est, dans ce cas, remise dans une plage correspondant à la largeur de la voie. On peut noter une légère anticipation (80ms) de l'algorithme due au fait que l'estimation est faite à hauteur du véhicule mais que la caméra observe à plusieurs mètres devant le véhicule.
D'une manière générale, on constate que la précision de positionnement latéral se situe (sous réserve d'un calibrage correct) dans un intervalle de +/- 5cm dans 95% des situations. Cette précision sera néanmoins dégradée en situations difficiles (peu de données observées, reconnaissance partielles des bandes, etc).
La Figure 4b montre, de la même manière, la comparaison de l'estimation de la courbure horizontale de la route. On constante là encore une légère anticipation (80ms), et de temps en temps une valeur légèrement sous-estimée de la courbure. Cette erreur est due aux hypothèses utilisées pour le passage 2D/3D. D'une manière générale, on observe une erreur résiduelle d'environ $0.0001m^{-1}$ en faibles courbures. Cette erreur peut monter à +/- 20% en fortes courbures (supérieures à $0.01m^{-1}$, la courbure maximale sur autoroute étant d'environ $0.004m^{-1}$

2.3 Suivi dynamique

2.3.1 Introduction

Malgré les bonnes performances (en termes de précision) de l'approche proposée, il existe des situations où les marquages restent difficiles à détecter, soit du fait de leur non-détection des soit de par leur absence. Dans ces situations, il est malgré tout nécessaire de fournir un résultat dans le but d'alimenter les fonctions d'aide à la conduite envisagées.
Dans cette version initiale, les résultats issus de l'image précédente ne sont pas exploités. En effet, la recherche de la route doit être réalisée dans une grande partie de l'image, ce qui est d'une part une perte de temps, et d'autre part, risque d'entraîner de mauvaises détections. Par ailleurs, du fait que la sortie du système est au final l'état du véhicule sur la voie (position latérale et angle de cap), des déficiences ponctuelles de la reconnaissance de la route dans l'image entraîne une perte de localisation préjudiciable aux fonctions d'aide à la conduite.

L'approche retenue par le LASMEA dans le cadre d'ARCOS a donc été de combiner l'utilisation de capteurs proprioceptifs afin à la fois de fournir un résultat de localisation probable du véhicule, mais aussi de permettre une meilleure focalisation de recherche des bords dans l'image suivante. En effet, l'absence de détection durant un temps faible ne devrait pas être préjudiciable dans la mesure où l'on peut extrapoler la trajectoire du véhicule grâce à ses capteurs embarqués.

Une phase de « suivi dynamique » s'est donc avérée nécessaire. Cette phase permet :
1.de prévoir au mieux quelles seront les positions des bords de la route dans l'image suivante, ceci afin à la fois de réduire les espaces de recherche (et donc les temps de calcul) mais aussi et surtout d'accroître la fiabilité du système (du fait que la recherche est plus localisée, la probabilité de fausse détection est fortement limitée),
2.de donner une information réaliste sur les paramètres de localisation recherchés même en l'absence ponctuelle de détection. En effet, dans ce cas, l'évolution / suivi qu'autorisent des capteurs proprioceptifs (qui donnent des informations sur l'état du véhicule, comme un gyromètre ou un odomètre par exemple) permet de « lisser » les manques ponctuels de mise à jour par les données image.
3.De gérer de manière élégante les changements de voie de circulation. En effet l'algorithme de détection de route dans l'image tente une détection de deux bords cohérents et conformes à une voie de circulation. Les changements de voies entraînaient donc des sauts dans la localisation.

Par ailleurs, il est très classique que les algorithmes de ce type fonctionnent selon deux modes: un mode d'initialisation, et un mode suivi. Le premier, à partir d'une analyse (souvent plus coûteuse) de toute l'image, fournit un premier résultat. Le mode suivi est ensuite enclenché avec l'hypothèse d'une faible évolution du véhicule par rapport à l'instant de la première image.
La difficulté de cette approche est que de temps en temps, des pertes de détection entraînent la nécessité de repasser en mode initialisation. L'établissement des critères de passage de l'un à

l'autre mode reste difficile à établir. Nous décrivons plus loin l'approche élégante que nous avons développée pour traiter ce problème.

Capteurs proprioceptifs

Afin d'alimenter ce système, il est nécessaire d'avoir ces informations "proprioceptives" caractérisant l'état du véhicule durant les phases de non-reconnaissance de la chaussée. Dans le cadre d'ARCOS, nous avons équipé notre véhicule d'un capteur odométrique et d'un capteur d'angle au volant (réalisé au LASMEA : voir Figure 5). Afin de maintenir une certaine modularité ces capteurs ont été interfacés sur un bus CAN, grâce à des modules électroniques développés au LASMEA.

Figure 5 - Montage mécanique du capteur d'angle au volant du véhicule VELAC

Le calibrage du capteur volant n'a cependant pas été simple. En effet, il est recherché ici l'information « angle des roues » afin d'extrapoler la trajectoire du véhicule (voir paragraphe 4.2.1.2).

Nous avons ensuite élaboré les modèles théoriques permettant la prise en compte de ces nouvelles informations et les structures logicielles qui doivent gérer l'arrivée désynchronisée de ces données.
L'objectif recherché ici est de donner une bonne estimation de la position du véhicule entre deux étapes de reconnaissance de la chaussée. On conçoit que plus une telle anticipation est correcte, moins les pertes de reconnaissance de la chaussée seront cruciales.
Après de nombreux essais, il s'est avéré que malgré un calibrage précis et soigné du capteur de l'angle au volant (mise en place du véhicule sur un banc d'analyse angulaire de la direction) les anticipations reposant sur cet angle n'ont pas donné de résultats probants sur toute la dynamique de courbure envisageable.
En effet notre capteur fournit la valeur angulaire du volant de la direction qui n'est pas homocinétique à la direction elle-même (une petite non-linéarité du système crémaillère et un joint mécanique de type cardan relient ces deux angles). Une estimation point par point a donc dû être faite. Malgré cela, il s'est avéré que le comportement du modèle, s'il est ajusté de manière à être correct pour les basses courbures, fournit de mauvaises extrapolations aux fortes courbures et réciproquement.

Nous avons constaté que le paramètre vitesse était important lui aussi. Du fait des roulis engendrés par la vitesse, pour de mêmes informations angulaires, des différences de trajectoires du véhicule ont été mesurées.
La prise en compte de ces divers effets nécessiterait l'utilisation d'un modèle dynamique du véhicule. De tels modèles sont bien connus mais ils nécessitent, à leur tour, une identification des paramètres intervenant dans la dynamique du véhicule, rendant le problème à la fois difficile mais surtout peu

reproductible pour d'autres véhicules(notre algorithme a été porté sur d'autres véhicules dans le cadre des fonctions ARCOS).

Forts de ces constats, nous avons intégré un nouveau capteur : un gyromètre (Gyrostar ENV05F-03 de muRata). Son étalonnage a été réalisé grâce à la plate-forme robotique Motoman du LASMEA qui a permis de vérifier sa bonne linéarité dans toute la gamme de vitesse angulaire envisagée pour l'application. Le capteur a ainsi pu être soumis à diverses contraintes dynamiques parfaitement contrôlées et reproductibles puis a ensuite été intégré dans notre véhicule et connecté lui aussi sur le bus CAN. Nous avons aussi établi les équations permettant de déduire l'évolution du véhicule. Il faut cependant noter que ce capteur semble sensible à la température, la prise en compte de ce point fera l'objet de travaux futurs.

Afin de comparer les résultats d'estimation obtenus tant pour l'angle volant que pour le gyromètre nous avons estimé la trajectoire du véhicule à partir d'une position initiale et une orientation connues. Afin d'avoir une « vérité terrain » nous avons également enregistré la valeur de la position du véhicule extraite du GPS (cependant peu précis : +/- 15m) de notre véhicule.

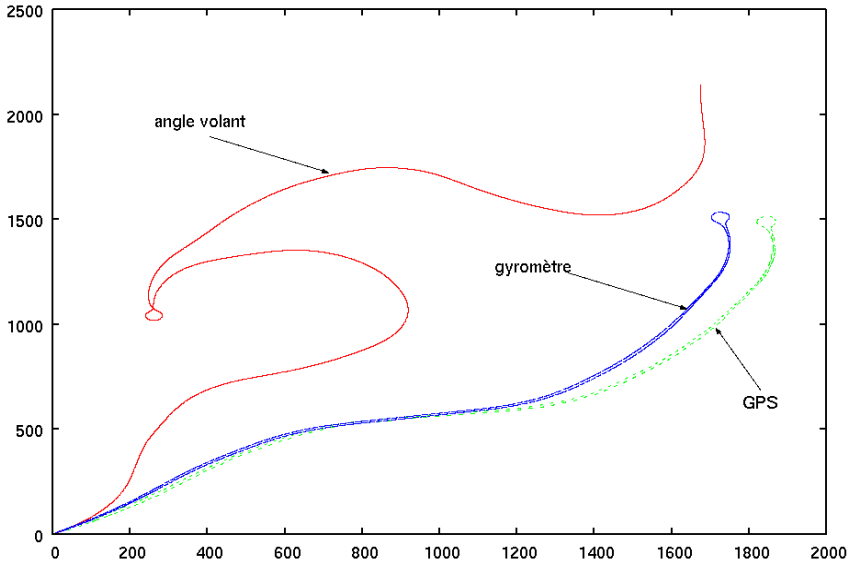


Figure 6 : estimation de la position du véhicule à partir d'une initialisation connue en utilisant l'angle volant, le gyromètre et un GPS.

Ces résultats sont représentés en Figure 6. La courbe GPS montre l'allure de la trajectoire utilisée : il s'agit d'un boulevard dans la périphérie de Clermont-Ferrand (la trajectoire utilisée fait environ 3Km) au bout duquel se trouve un rond point permettant de tester de grande disparités de courbure. On voit que l'angle volant ne fournit une estimation fiable que sur quelques mètres (un calibrage permettrait de le faire répondre mieux dans les faibles courbures mais au détriment de la qualité dans les fortes courbures). Le gyromètre s'avère, par contre très performant.
La Figure 7 montre, quant à elle, le comportement des capteurs dans le rond point (l'initialisation est dans ce cas faite en entrée du rond-point). Le véhicule a réalisé trois tours. On voit que l'angle volant engendre une erreur de cap de plus de 90 degré en sortie de rond point, contrairement au gyromètre qui lui, fournit une estimation très réaliste. La petite excentration entre GPS et gyromètre provient, à la fois de fait que ces deux capteurs ne sont pas situés au même endroit dans le véhicule mais aussi à cause de l'erreur du GPS qui, à cette échelle n'est plus négligeable.

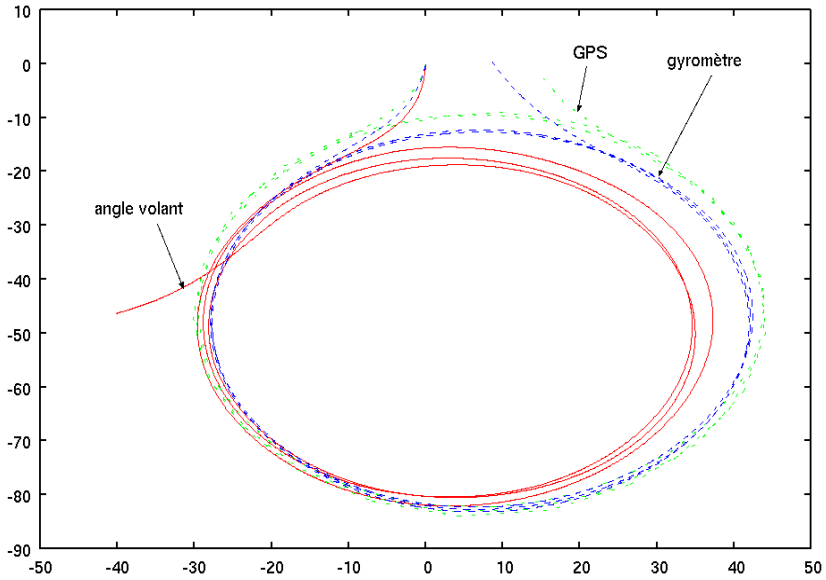


Figure 7 - Estimation de la position du véhicule dans le rond-point.

On peut donc remarquer que le gyromètre fournit une estimation fiable dans toute la plage de courbure observée. Par ailleurs, la mise en place d'un tel capteur s'est avérée beaucoup moins problématique comparativement à l'installation du capteur sur le volant de notre véhicule. C'est donc ce capteur qui a été installé pour l'algorithme utilisé.

2.3.2 Suivi dynamique

Le principe du suivi dynamique est de réduire l'intervalle de recherche des bords dans l'image k+1 à partir des estimations faites à l'image k, mais aussi à partir des données proprioceptives entre les deux instants, et leur imprécision. Dans l'hypothèse où la détection à l'image k+1 a échoué, une nouvelle extrapolation pour l'image k+2 sera faite avec, naturellement une imprécision qui augmente. Dans le cas où un nombre important des non-détections est survenu, l'imprécision atteint asymptotiquement la valeur d'apprentissage initiale utilisée pour la reconnaissance de la route dans la première image.
Nous voyons donc apparaître ici l'approche unifiée dans laquelle initialisation et suivi sont confondues. La position du véhicule, mise à jour par la phase de localisation (réalisée après chaque détection de la route dans l'image) est extrapolée pour l'instant suivant grâce à l'utilisation des capteurs proprioceptifs (odomètre et gyromètre) selon un modèle cinématique du véhicule (modèle d'Ackermann). Cette extrapolation permet à la fois de prédire l'état tridimensionnel (position, orientation, etc.) du véhicule mais aussi l'intervalle de confiance sur celui-ci. Cet intervalle permet ensuite de fixer la taille des zones de recherche pour la nouvelle image, réalisant ainsi le suivi dynamique.

La validation de cette approche s'est faite pour l'instant de manière purement qualitative. En effet, à présent, l'algorithme fournit toujours une solution (c'est-à-dire une estimation de sa position latérale, de son angle de cap, de la largeur de la chaussée, de la courbure la route à l'avant). L'algorithme fournit néanmoins une matrice de covariance caractérisant la confiance que l'on peut donner à ces valeurs à chaque instant.

D'une manière subjective, le fonctionnement est très correct (voir figure 8). Lorsque l'on se trouve en situation de non reconnaissance (on voit ici le cas où le conducteur effectue un changement de voie, la faible taille du bord gauche de la voie interdit la reconnaissance). On peut alors constater que le modèle extrapolé dans l'image se trouve relativement bien superposé aux voies réelles. Cette situation a pu aussi être recréée de manière artificielle (l'opérateur place sa main devant la caméra et effectue des manoeuvres. La bonne superposition des voies reconstruites montre la validité de l'approche.

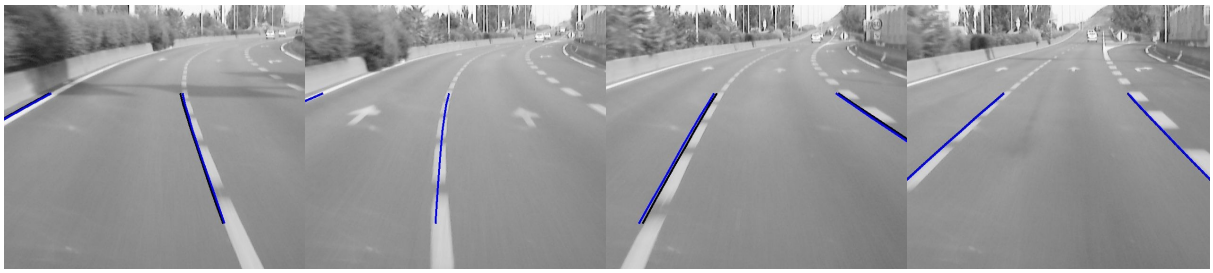


Figure 8 - Résultats de la reconnaissance / suivi dans le cas d'un changement de voie

Il faut cependant noter que même si un suivi proprioceptif peut être simple et de très bonne qualité dans un repère absolu, celui-ci n'est pas simple en relatif (selon un repère lié au véhicule). En effet, un tel suivi a pour objectif d'anticiper à chaque instant la position du véhicule sur la voie (avec notamment sa position latérale x0 par rapport aux bords, l'angle de cap ψ , qui est l'angle que fait l'axe du véhicule par rapport à la tangente au bord et la courbure de la voie). On voit, sur la Figure 8, que l'anticipation de ces paramètres pour l'image suivante (notamment l'angle ψ) nécessitera une bonne connaissance de la courbure de la voie. Or ce paramètre, déduit de l'algorithme peut, d'une part être sujet à erreur, et d'autre part, dans le cas où la détection de la chaussée a échoué, ne sera que simplement estimé (par un modèle de type clothoïdal). Si ce type d'erreur ne pose pas de problème en ligne droite, on pourra cependant observer des imprécisions de localisation en virage dans les cas de non-reconnaissance de la voie sur plusieurs images.

Ces résultats doivent cependant être validés de manière quantitative. Pour cela, il est nécessaire d'avoir un GPS très précis afin de comparer les trajectoires réelle et extrapolées du véhicule durant les phases de non-reconnaissance.
Nous avons acquis récemment un tel capteur (GPS Omnistar avec correction différentielle satellitaire de précision supérieure 40cm à un écart-type), mais il ne nous a pas été possible à ce jour de l'interfacer au sein de notre véhicule.

2.3.3 Tests ARCOS

Des tests ARCOS de novembre 2003 et mars 2004 ont été réalisés afin de caractériser les algorithmes. Tous les algorithmes n'étant pas en mesure de fournir les mêmes données, il est difficile de les comparer de manière objectives. Cependant, l'algorithme du LASMEA a fonctionné de manière parfaite sur les données de synthèse fournies par le LIVIC, les résultats de localisation étant ceux décrits précédemment. Par ailleurs les tests ont aussi été réalisés sur des images réelles (sur une banque de séquences communes à tous). Le fonctionnement a là aussi été très satisfaisant puisque l'algorithme a été évalué comme étant le « plus performant et le plus robuste ».
Il est cependant nécessaire de remarquer que la phase de « suivi » de l'algorithme était, durant ces tests, réduite à sa plus simple expression puisque les données des capteurs proprioceptifs n'avaient pas été prises en compte ni dans les données de simulation ni dans les séquences réelles.

2.4 Portage de l'algorithme dans de cadre des fonctions ARCOS

Dans le cadre des travaux d'ARCOS, un certain nombre de fonctions ont été portées sur les véhicules de test (parmi lesquels : ceux du LIVIC, de l'Ecole des Mines de Paris, du l'UTC de compiègne, et du LASMEA : voir figure 9). La fonction « prévention des sorties de route » a ainsi été portée sur les véhicules du LIVIC et de l'UTC. L'algorithme présenté ici était ainsi au coeur de cette fonction mais aussi de la « gestion des interdistances » qui fait l'objet du présent rapport.

Figure 9 - Le véhicule VELAC du LASMEA

L'ensemble du code a été livré tant au LIVIC qu'à l'UTC (après signature de conventions). Une procédure de calibrage a été développée afin de permettre une mise en oeuvre efficace.

Il est à noter qu'à la suite de tests réalisés durant l'année 2004, l'algorithme de reconnaissance des voies, tel qu'il a été présenté ici a montré quelques faiblesses sur les phases de détection de bas niveau (en particulier, pour la détection détection des parties de marquage peu contrastées). L'algorithme du LIVIC, quant à lui, a montré de bonnes performances de bas niveau mais un fort manque de cohérence géométrique entre les deux bords (un bord pouvait montrer, sur une même image une courbure à droite, et l'autre une courbure à gauche). Une collaboration a donc été entamée avec le LIVIC pour essayer de combiner les algorithmes des deux laboratoires. Pour ce faire, la structure de haut niveau de notre algorithme est utilisée pour focaliser la recherche dans l'image des bords de la route. Une fois les bords de la route retrouvés, la localisation du véhicule par l'approche du LASMEA sera ensuite exploitée. La phase de suivi dynamique utilisant les capteurs proprioceptifs, est naturellement maintenue. Cette recherche a fait l'objet de présentations dans le cadre de la journée du 28 octobre 2004 à Satory. Cette combinaison a été très fructueuse puisque la collaboration continue et un article en commun a été soumis au congrés Machine Vision and Application 2005.

2.5 Bibliographie

[1] Potelle A. , Chapuis R. Surveillance temps réel de trajectoire sur autoroute. Technique et Science Informatique, 16(1) :3968, 1997.
[2] Aufrère R. Reconnaissance et suivi de route par vision artificielle, application à l'aide à la conduite. Thèse de l'Université Blaise Pascal Clermont-Ferrand (France), Juin 2001.
[3] Aufrère R., Chapuis R., Chausse F. A dynamic vision algorithm to locate a vehicle on a non-structured road. International Journal of Robotics Research, 19(5) :411423, Mai 2000.
[4] Aufrère R., Chapuis R., Chausse F. Détection précise de bords de route par vision monoculaire embarquée. In 12ème congrés RFIA, volume II, pages 229-238, Paris (France), 13 Février 2000.
[5] Aufrère R., Chapuis R., Chausse F. A fast and robust vision based road following algorithm. In IV'2000 (IEEE Int. Conf. on Intelligent Vehicles), pages 192-197, DearBorn (MI USA), Octobre 16 2000.
[6] Aufrère R., Chapuis R., Chausse F. A model driven approach for real time road recognition. International Journal of Machine Vision and Applications, 13 :95107, Décembre 2001.

[7] Chapuis R., Aufrère R., Chausse F., Alizon J. A fast and robust vision based road following algorithm. IV'2001 (IEEE Int. Conf. on Intelligent Vehicles), pages 1318, Tokyo (Japon), Mai 1417 2001.
[8] Chapuis R., Aufrère R., Chausse F. Accurate road following and reconstruction by computer vision. IEEE Trans. on Intelligent Transportation Systems, 3(4), Décembre 2002.

N. TRUJILLO, F. CHAUSSE, R. CHAPUIS, C. BLANC, On Road Simultaneous Vehicle Recognition and Localization by Model Based Focused Vision, IAPR Conference on Machine Vision Applications, May 16-18, Tsukuba Science City, 2005.

J. DOURET; R. LABEYRADE,; J. LANEURIT; R. CHAPUIS, A reliable and robust lane detection system based on the parallel use of three algorithms for driving safety assistance., IAPR Conference on Machine Vision Applications, May 16-18, Tsukuba Science City, 2005.

J. LANEURIT, R. CHAPUIS, F. CHAUSSE, *Accurate vehicle positioning on a numerical map,* International Journal of Control Automation and Systems, v.3, n. 1, pp. 15-31, Mars 2005.

3 Détection d'obstacles par télémétrie laser à balayage

3.1 Le capteur LMSZ21060

3.1.1 Présentation

Le télémètre Laser 3D LMSZ210-60 (voir Figure 10) est un système capable de délivrer des images 3D. Ces images sont basées sur la mesure de distance précise délivrée par un système opto-électronique et sur un mécanisme à double balayage. Elles sont formées en exécutant une série de mesure de distance dans des positions différentes, avec des directions angulaires bien définies. Ces données de distance associées aux angles forment la base des images 3D. Ces images 3D ou de profondeur se présentent sous la forme de matrice en analogie avec les images de luminance. Chaque élément de la matrice représente un pixel. Chaque pixel donne une information sur la scène observée par le capteur 3D. Cette information représente une indication géométrique de la scène. En résumé, l'information contenue par chaque pixel de l'image de profondeur représente les coordonnées géométriques de celui-ci dans le repère du capteur.

Figure 10. Le télémètre laser LMS-Z210-60

3.1.1.1 Le module de mesure de distance

Le système de mesure de distance est basé sur le principe du calcul de temps de vol des impulsions laser de longueur d'onde dans le proche infrarouge.

Un générateur d'impulsions électriques commande une diode laser semi-conductrice qui envoie des impulsions de lumière infrarouge vers un objectif émetteur. A travers l'objectif récepteur, une partie du signal réfléchie par une cible frappe une photodiode, ce qui génère un signal électrique. L'intervalle de temps entre l'émission et la réception est mesuré au moyen d'une fréquence d'horloge stabilisée par un quartz. La mesure de distance calculée est transmise à un microprocesseur interne qui prépare les données à transmettre au PC (voir Figure 11).

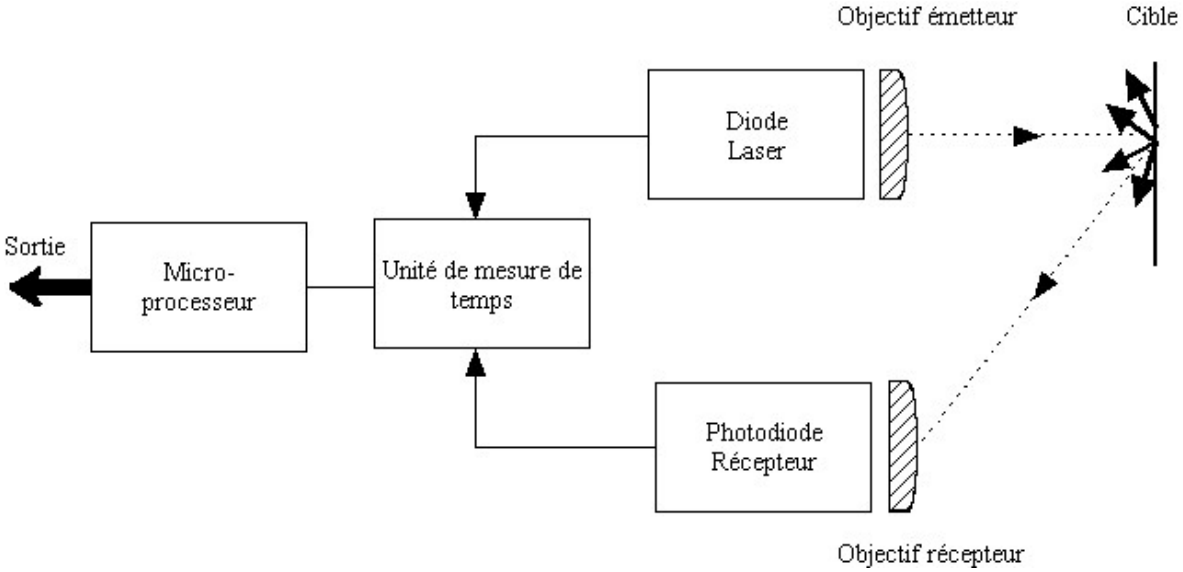


Figure 11. Principe de mesure de distance

3.1.1.2 Le module de balayage

La tâche du mécanisme de balayage est de diriger le faisceau laser dans une position précisément définie. Les images 3D (appelée aussi frame) sont composées de lignes, et chaque ligne est composée de pixels. Ce mécanisme fournit un balayage rapide pour les lignes (typiquement 20 lignes par seconde), et plusieurs centaines de pixels par ligne. Le mécanisme de balayage pour les frames est beaucoup plus lent que celui pour les lignes qui contiennent typiquement des centaines de pixels.
La déflection angulaire du faisceau laser est réalisée par un miroir polygonal rotatif (voir Figure 12). Les miroirs polygonaux sont habituellement en rotation à vitesse constante afin de fournir des balayages répétitifs unidirectionnels. Le nombre de facettes du polygone rotatif fixe l'amplitude de l'angle de déflection. Notre laser est équipé d'un miroir polygonal rotatif à quatre facettes (60°). Un faisceau laser à diamètre constant est divisé au niveau de l'arrête du polygone en deux faisceaux se propageant dans différentes directions. Par conséquent, le temps total d'une révolution complète ne pourra pas être utilisé pour le balayage et le secteur utilisable de balayage doit être réduit.

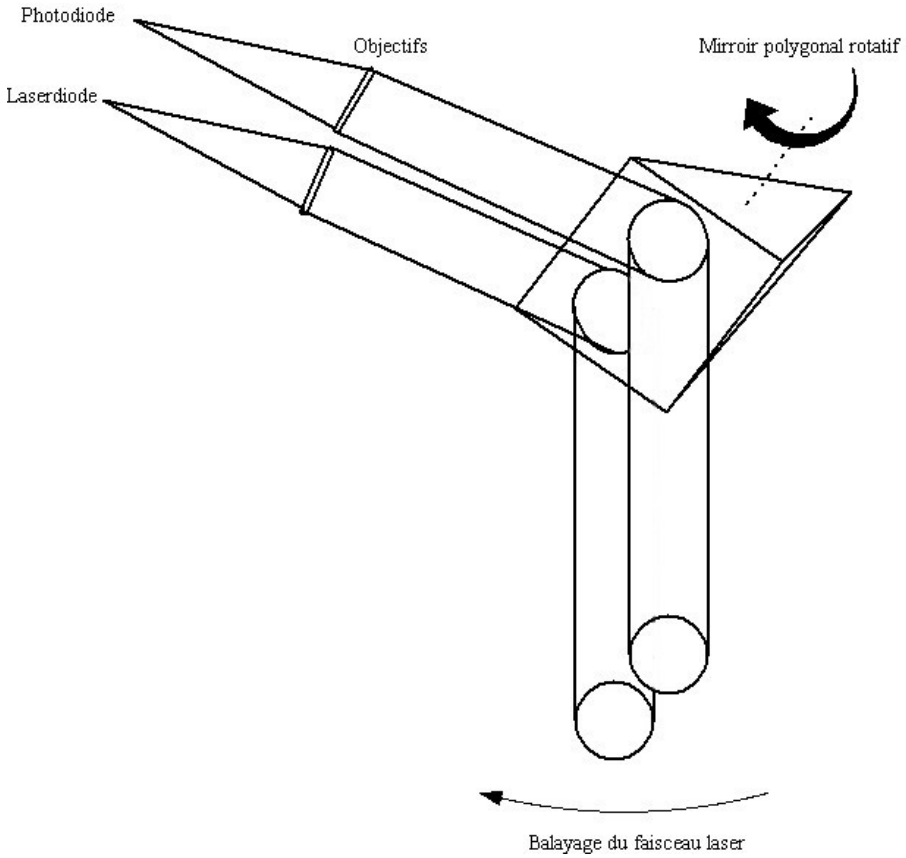


Figure 12. Principe du balayage du faisceau laser avec un miroir rotatif

Le mécanisme de balayage formant les images, plus lent que celui des lignes, se fonde sur la rotation du système formé par la tête optique et le balayage des lignes. Ceci est accompli en montant le système de balayage des lignes et la tête optique sur un système tournant.

3.1.2 Spécifications du capteur

3.1.2.1 Performances et caractéristiques du capteur

Les performances et caractéristiques du capteur sont résumées dans les tableaux suivants :

Mécanisme de balayage	Miroir polygonal rotatif
Nombre de facettes	4
Angle de balayage	60° fixé
Mouvement angulaire	Linéaire, unidirectionnel
Vitesse de balayage	5 balayages/s → 90 balayages/s
Pas angulaire	0,072° → 0,57°
Résolution angulaire	0,036°

Tableau 3-1 - Caractéristiques de balayage des lignes

Mécanisme de balayage	Tête
Angle de balayage	0°→333° (fixé)
Mouvement angulaire	Linéaire
Vitesse de balayage	1°/s → 20°/s
Pas angulaire	0,072° → 0,36°
Résolution angulaire	0,018°

Tableau 3-2 - Caractéristiques de balayage des lignes

Mesure de distance	0,5 m → 150 m
Résolution	25mm
Précision	2,5 cm
Cadence des tirs	→ 28 kHz
Longueur d'onde	Proche infrarouge 0,9 µm
Divergence du faisceau	3 mrad
Sécurité laser	Classe 1

Tableau 3-3 - Caractéristiques de balayage des lignes

Ces caractéristiques sont adaptées à notre application. En effet, la gamme de mesure est appropriée à la détection d'obstacles routiers et la divergence permet d'obtenir une résolution satisfaisante (1 mrad correspond à un impact de 10 cm de diamètre à 100 m).

3.1.2.2 Définitions des axes

La Figure 13 décrit le repère absolu lié au capteur qui sera utilisé dans la suite.

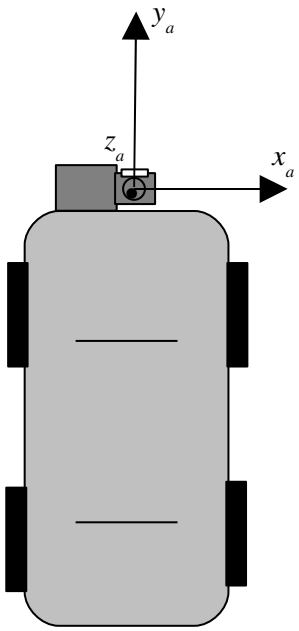


Figure 13 - Le repère lié au véhicule

3.1.2.3 Caractéristiques temporelles

Les caractéristiques temporelles du capteur correspondent au temps nécessaire au capteur pour acquérir une image 3D de la scène. Ce temps est lié à la vitesse de rotation maximale de la tête optique (20°/s) et à la vitesse de rotation maximale du miroir polygonal rotatif à quatre facettes (90 lignes/s soit 90/4=22,5 tours/s). Ces vitesses fixent donc une contrainte sur le pas angulaire à ne pas dépasser pour le balayage des lignes. Le temps nécessaire pour acquérir une image 3D complète est l'angle total fixé par le nombre de lignes divisé par la vitesse angulaire de la tête optique.
Déterminons le temps nécessaire pour acquérir une image 3D de 100 pixels par 20 lignes le plus rapidement possible. Ce type d'images 3D sera le type d'images utilisé dans notre système de perception. Ses dimensions correspondent, après différents tests, aux dimensions minimales en vue d'une détection des obstacles aisée. Le Tableau 3-4 présente les différents paramètres du capteur à régler (en jaune) ainsi que les contraintes. De plus, suivant ces différentes données, le temps d'acquisition théorique est donné. On remarque que le temps réel d'acquisition est supérieur au temps d'acquisition théorique. Cette supériorité est due au temps que le moteur de la tête optique met pour réaliser un aller-retour et prend donc en compte l'enchaînement des images. C'est une contrainte mécanique impossible à supprimer.

		Contraintes
nombre de lignes/images	20,000	
angle total pour une image	4650 mrad	
pas angulaire entre deux lignes	244,7 mrad	<245,5 mrad
nombre de pixels/ligne	100	
angle total pour une ligne	63360 mrad	
pas angulaire entre deux pixels	640,0 mrad	
vitesse du polygone rotatif	22,4 tours/s	<22,5 tours/s
nombre de lignes/sec	89,6	90
vitesse angulaire de la tête optique	21,9 rad/s	22,2 rad/s
	19,7 °/s	20 °/s
temps d'acquisition théorique	212 ms	
Temps réel d'acquisition	proche de 550 ms	

Tableau 3-4 - Caractéristiques temporelles pour des images de 100 pixels par 20 lignes

3.1.2.4 Images 3D ou de profondeur

Ici, nous montrons différents types d'images 3D en insistant sur les images utilisées dans notre système de détection d'obstacle.
Les premières images correspondent à une image dense d'un bureau. Elles correspondent à une rotation de la tête optique de 333° et un angle de balayage de 60° (fixe).

Figure 14. Image d'intensité dense : un bureau

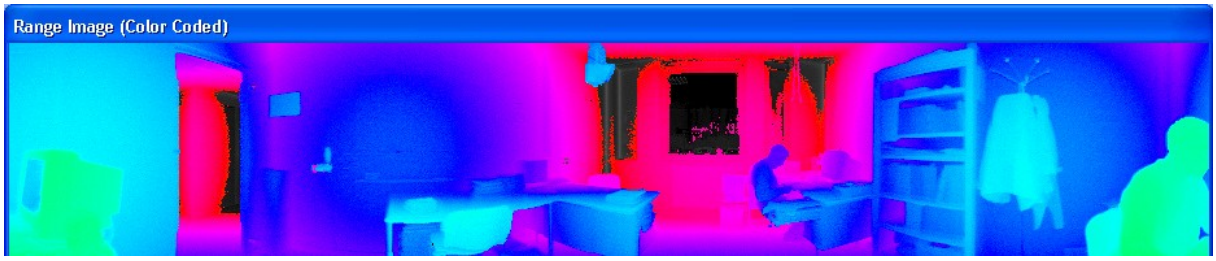


Figure 15. Image 3D dense : un bureau

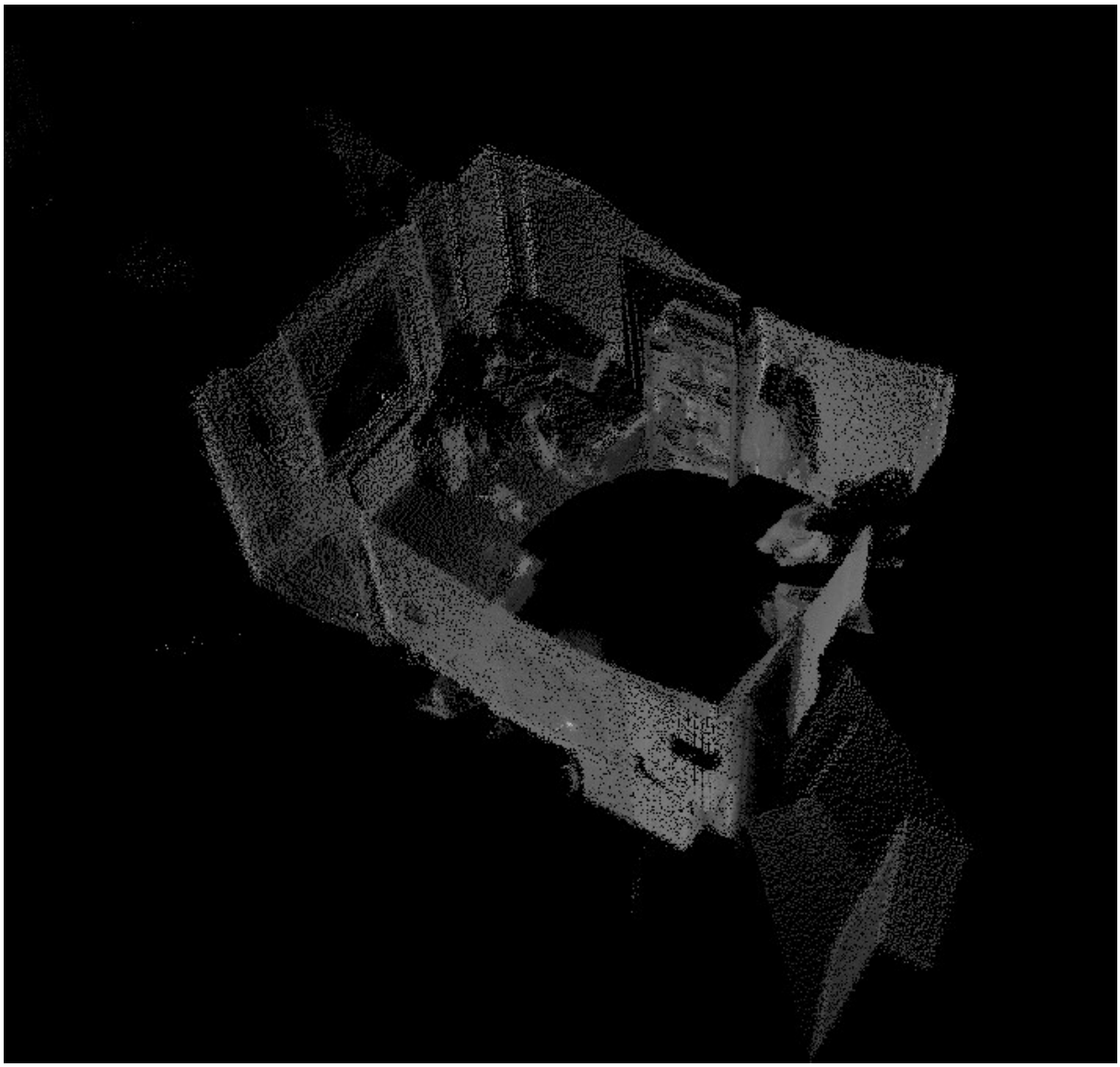


Figure 16. Une vue 3d du bureau au format wrl

La Figure 14 est une représentation de l'image 3D par code couleur (couleurs différentes par classe de distance). La Figure 15 correspond à l'image d'intensité laser reçue par le capteur. L'image en 3 dimensions (Figure 16) permet de tourner autour d'un objet avec la souris, de le déplacer, de s'en rapprocher, comme s'il était réel. Elle permet aussi la vision panoramique, par exemple d'un paysage ou la visite virtuelle d'une maison ou d'une auto.

Nous présentons maintenant différentes images utilisées dans notre système de perception. Ces images dont les dimensions sont de 100 pixels par 20 lignes sont zoomées quatre fois pour une meilleure compréhension de la scène. La dimension des images est fixée par les caractéristiques temporelles du capteur qui seront présentées dans la suite. Nous montrons ici différents types d'images correspondant à plusieurs types d'obstacles (voitures : Figure 17; camions : Figure 18, motos : Figure 19, piétons : Figure 20). Pour toutes ces figures, l'image du haut correspond à l'image d'intensité laser retournée par le capteur, l'image du milieu représente l'image 3D avec un code couleur (la couleur dépend de la distance de chaque point 3D) et enfin l'image du bas représente la vue de la scène.

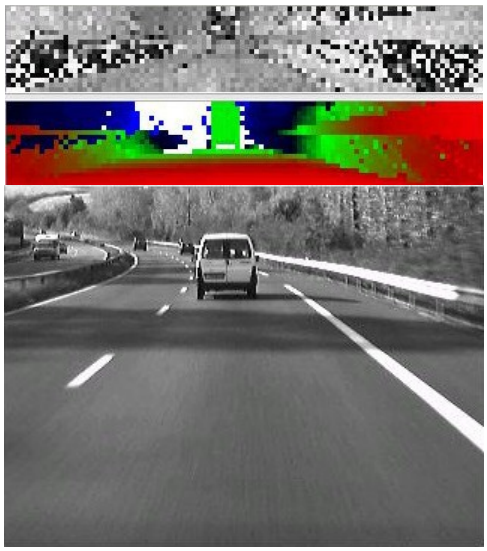


Figure 17. Visualisation d'une voiture à 30 m

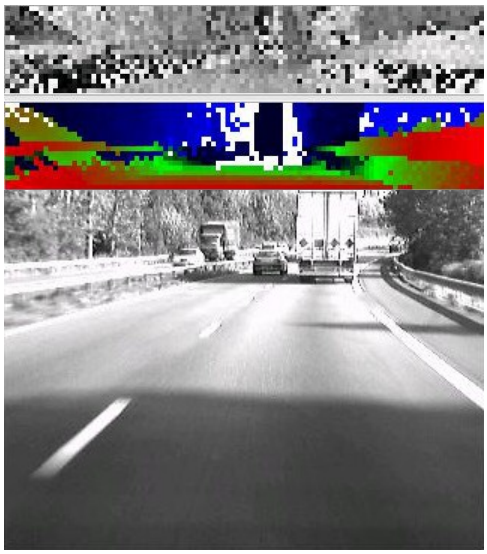


Figure 18. Visualisation d'un camion à 50 m

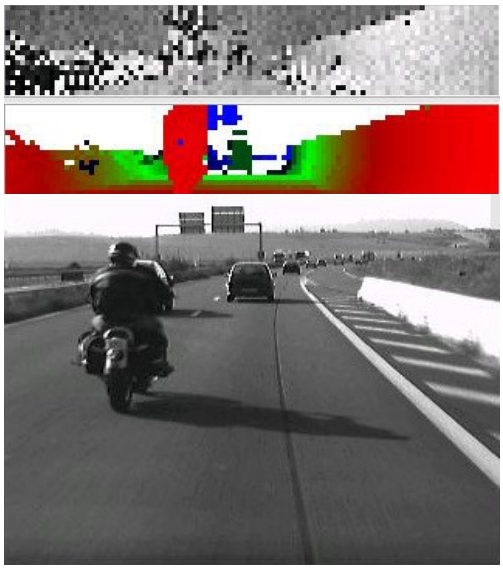


Figure 19. Visualisation d'une moto à 10 m

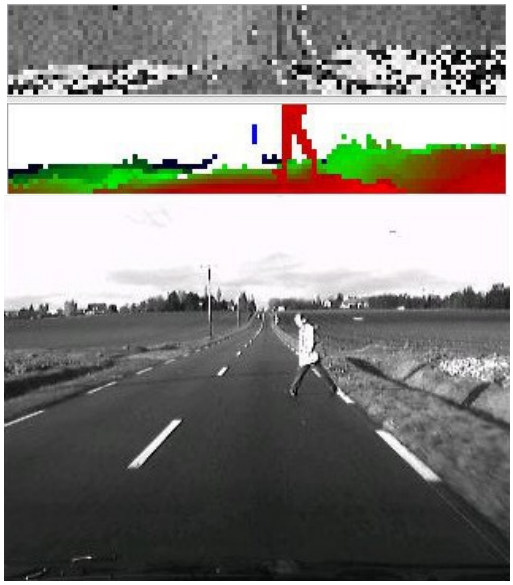


Figure 20. Visualisation d'un piéton à 11 m

Ces différentes images montrent premièrement qu'il n'est pas possible d'utiliser l'intensité laser reçue par le capteur en vue de la détection d'obstacle. Cette donnée ne sera donc pas utilisée dans la suite. Ensuite, on remarque qu'il existe une assez bonne discrimination des obstacles dans les images de profondeur ce qui permettra une détection de ceux ci. Cependant, cette détection sera d'autant plus difficile que la distance des obstacles augmente et qu'ils sont de faibles dimensions.

3.2 La détection d'obstacle par laser

La détection d'obstacle correspond à l'extraction de primitives dans les images 3D présentées dans la partie précédente. Cette extraction se déroule en deux étapes [Trass93] . La première étape consiste en la segmentation en régions de l'image 3D et la deuxième étape consiste à reconnaître des obstacles parmi ces régions.

3.2.1 Segmentation en régions de l'image 3D

L'analyse des images de profondeur est couramment réalisée grâce à des méthodes de segmentation. Ces méthodes bas niveau consistent à regrouper ensemble les pixels présentant des propriétés similaires. De nombreuses méthodes existent dans la littérature. Une bibliographie est présentée dans [Chec 96]. Pour notre système de perception, le but de la segmentation est d'obtenir des zones formées d'impacts laser spatialement proches. Le critère de segmentation est la distance car, dans l'hypothèse d'une route plane, les obstacles routiers apparaissent comme des plans orthogonaux à la route constitués de points situés à une distance y_a constante à une tolérance σ_y près. La méthode retenue pour cette segmentation s'appuie sur un algorithme de croissance de régions. Les régions sont construites à partir d'un impact et croissent en incorporant, de proche en proche, les impacts situés à la même distance y_a de notre véhicule avec une tolérance σ_y. Une description du principe de l'algorithme est donnée ci-dessous.

```
tous les impacts sont non classés
Pour chaque impact de l'image
        Si distance non nulle et impact non classé
        initialisation d'une nouvelle région
        impact classé
        Pour chacun des voisins
                Si distance du voisin égale à près et impact non classé
                        placer le voisin dans la région
                        placer le voisin dans le tampon de recherche
                        voisin classé
                FinSi
                Tantque le tampon de recherche n'est pas vide
                        Pour chacun des voisins de l'impact du tampon
                                Si distance du voisin égale à près et impact non classé
                                        placer le voisin dans la région
                                        placer le voisin dans le tampon de recherche
                                        voisin classé
                                FinSi
                                éliminer l'impact du tampon de recherche
                        FinPour
                FinTantque
        FinPour
        FinSi
FinPour
```

Finalement, après la segmentation un ensemble de régions R_i comportant N_i impacts est généré. Parmi ces régions, toutes ne correspondent pas forcément à un obstacle. Il est donc nécessaire d'effectuer une étape de reconnaissance pour ne retenir que les régions correspondant à des obstacles potentiels. Un exemple de segmentation est représenté sur la figure 16.

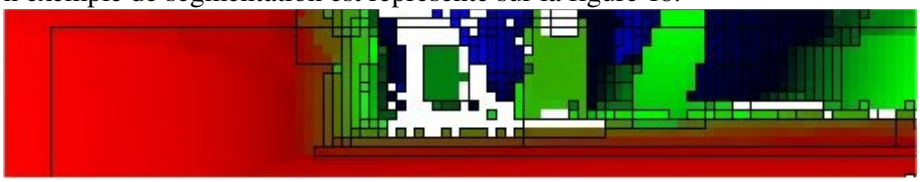


Figure 21. Résultats de segmentation

3.2.2 Reconnaissance de l'obstacle

La reconnaissance consiste donc à extraire des régions issues de la segmentation les obstacles potentiels. Par exemple, il faudra extraire les deux obstacles de type voiture présents dans la scène représentée figure 16. Nous limiterons notre reconnaissance dans un premier temps à la recherche d'obstacles de type voiture ou camion qui sont évidemment les plus courants dans un contexte routier. Cette reconnaissance s'appuie sur le modèle 3D d'un véhicule vue de derrière qui est un simple rectangle. Il s'agira donc de comparer les dimensions de chaque région à celle du modèle ainsi défini afin de prendre une décision quant à l'assimilation d'une région à un obstacle.
Chaque région R_i issue de la segmentation comportant N_i impacts est paramétrée par un vecteur :

$$R_i = (X_{\min,i}, X_{\max,i}, Z_{\min,i}, Z_{\max,i}, Y_i)$$

où $(X_{\min,i}, X_{\max,i})$ et $(Z_{\min,i}, Z_{\max,i})$ sont respectivement les valeurs minimale et maximale des coordonnées des impacts suivant l'axe x et l'axe z du repère du capteur, et Y_i la distance de la région à notre capteur.

Ces paramètres sont calculés de la façon suivante :

$$X_{\min,i} = \min\{x_{a,i,j}, \forall j \quad (1,...,N_i)\}$$
$$X_{\max,i} = \max\{x_{a,i,j}, \forall j \quad (1,...,N_i)\}$$
$$Z_{\min,i} = \min\{z_{a,i,j}, \forall j \quad (1,...,N_i)\}$$
$$Z_{\max,i} = \max\{z_{a,i,j}, \forall j \quad (1,...,N_i)\}$$
$$Y_i = \sum_{j=1}^{N_i} y_{a,i,j} / N_i$$

Finalement, ces paramètres permettent le calcul de trois caractéristiques importantes pour la reconnaissance des obstacles :

- la largeur l_i : $l_i = X_{\max,i} - X_{\min,i}$
- la hauteur h_i : $h_i = Z_{\max,i} - Z_{\min,i}$
- la position du centre $C_i(X_{m,i}, Y_i, Z_{m,i})$:

$$X_{m,i} = (X_{\min,i} + X_{\max,i})/2$$
$$Z_{m,i} = (Z_{\min,i} + Z_{\max,i})/2$$

Une région R_i est finalement caractérisée par le vecteur suivant :

$$R_i = (l_i, h_i, X_{m,i}, Y_i, Z_{m,i})$$

En comparant les composantes de ce vecteur aux dimensions des modèles de véhicules moyennant certaines tolérances, nous pourrons assimiler une région à un obstacle.
Soit Ω l'ensemble des N obstacles routiers. Un obstacle est représenté par le vecteur $O_k = (L_k, H_k)$ où L_k et H_k sont respectivement sa largeur et sa hauteur. Pour chaque région R_i, on peut écrire :

$$R_i \; \Omega \qquad \begin{matrix} l_i = L_k & \sigma_{L_k} \\ h_i = H_k & \sigma_{H_k} \end{matrix} \quad \forall \quad k \quad (1,...,N)$$

où σ_{L_k} et σ_{H_k} sont les tolérances sur les dimensions de l'obstacle O_k qui tiennent compte des variations de tailles possibles pour une même catégorie de véhicules. La figure 17 représente le résultat de cette reconnaissance : seul les deux régions caractérisant les voitures sont retenues en tant qu'obstacle.

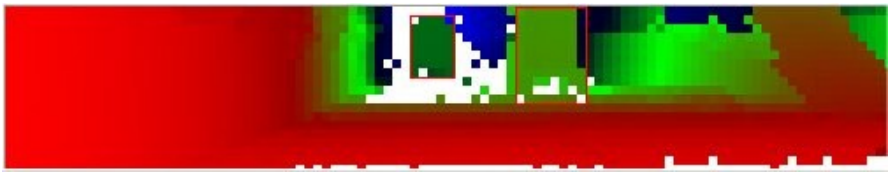


Figure 22. Reconnaissance des obstacles de type voiture

Cependant cette méthode de reconnaissance retourne quelques fausses détections dans différents cas illustrés dans les figures ci-dessous.

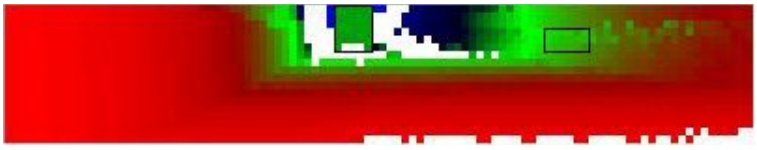


Figure 23. Des fausses détections

Pour éliminer ces fausses détections, des contraintes sur les normales aux différentes surfaces formant les régions (voir figure 19) sont ajoutées :

$$\theta_{x,i} = \Theta_{x,k} \quad \sigma_{\Theta_{x,k}}$$

$$\theta_{y,i} = \Theta_{y,k} \quad \sigma_{\Theta_{y,k}}$$

$$\theta_{z,i} = \Theta_{z,k} \quad \sigma_{\Theta_{z,k}}$$

où $\Theta_{x,k}$, $\Theta_{y,k}$, $\Theta_{z,k}$ représentent les angles de la normale à la surface d'un obstacle O_k , $\sigma_{\Theta_{x,k}}$, $\sigma_{\Theta_{y,k}}$, $\sigma_{\Theta_{z,k}}$ les tolérances sur ces angles, et $\theta_{x,i}$, $\theta_{y,i}$, $\theta_{z,i}$ les angles de la normale à la surface formée par la région R_i détectée.

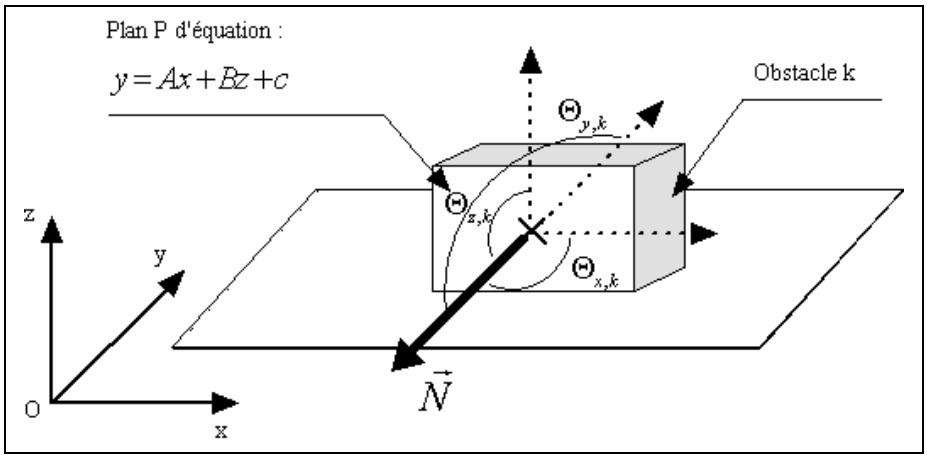


Figure 24. Représentation d'une normale à un obstacle

Cette méthode s'appuie sur l'approximation, au sens des moindres carrés, de l'ensemble des points de mesures formant la région $\left\{ P_i = (x_{a,i,j}, y_{a,i,j}, z_{a,i,j}) \forall j \quad (1...N_i) \right\}$ par un plan d'équation :

$$y_{a,i} = f(x_{a,i}, z_{a,i}) = a_i x_{a,i} + b_i z_{a,i} + c_i .$$

On calcul les coefficients (a_i, b_i, c_i) en minimisant le critère :

$$= \sum_{j=0}^{N_i} (y_{a,i,j} \quad a_i x_{a,i,j} \quad b_i z_{a,i,j} \quad c_i)^2$$

On a donc :

$$\begin{matrix} a_i \\ b_i \\ c_i \end{matrix} = M_i^{-1} Q_i$$

avec :

$$M_i = \begin{matrix} \sum_{j=1}^{N_i} x_{a,i,j}^2 & \sum_{j=1}^{N_i} x_{a,i,j} z_{a,i,j} & \sum_{j=1}^{N_i} x_{a,i,j} \\ \sum_{j=1}^{N_i} x_{a,i,j} z_{a,i,j} & \sum_{j=1}^{N_i} z_{a,i,j}^2 & \sum_{j=1}^{N_i} z_{a,i,j} \\ \sum_{j=1}^{N_i} x_{a,i,j} & \sum_{j=1}^{N_i} z_{a,i,j} & \sum_{j=1}^{N_i} 1 \end{matrix} \quad \text{et} \quad Q_i = \begin{matrix} \sum_{j=1}^{N_i} x_{a,i,j} y_{a,i,j} \\ \sum_{j=1}^{N_i} y_{a,i,j} z_{a,i,j} \\ \sum_{j=1}^{N_i} y_{a,i,j} \end{matrix}$$

On sait que le vecteur normal N_i à une surface en P_i, point de coordonnées $(x_{a,i}, f(x_{a,i}, z_{a,i}), z_{a,i})$, est donnée par :

$$\vec{N}_i = \frac{OP_i}{x} \quad \frac{OP_i}{z} = \begin{matrix} 1 \\ f_x(x_{a,i}, z_{a,i}) \\ 0 \end{matrix} \quad \begin{matrix} 0 \\ f_z(x_{a,i}, z_{a,i}) \\ 1 \end{matrix} = \begin{matrix} a_i \\ 1 \\ b_i \end{matrix}$$

où $f_x(x_{a,i}, z_{a,i})$ et $f_z(x_{a,i}, z_{a,i})$ sont respectivement les dérivées partielles en P_i de la fonction $y_{a,i} = f(x_{a,i}, z_{a,i})$ selon x et z.

Finalement, on en déduit facilement que :

$$\theta_{x,i} = \arccos \frac{a_i}{\left\| \vec{N}_i \right\|} \quad (\pi)$$

$$\theta_{y,i} = \arccos \frac{1}{\left\| \vec{N}_i \right\|} \quad (\pi)$$

$$\theta_{z,i} = \arccos \frac{b_i}{\left\| \vec{N}_i \right\|} \quad (\pi)$$

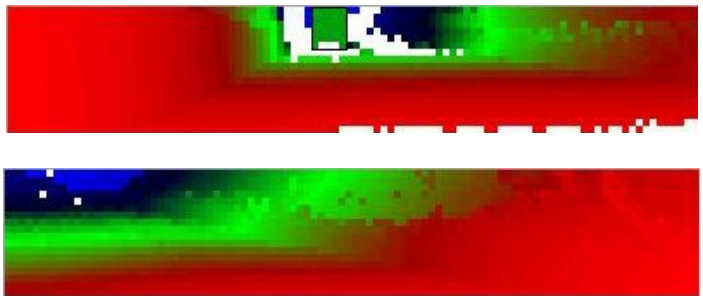


Figure 25. Résultats après ajout de contraintes sur les normales

Nous avons ainsi détecté les obstacles et éliminer les fausses détections (voir figure 20), il est maintenant nécessaire de mettre en œuvre leur suivi afin d'en extraire des informations cinématiques importantes.

3.3 Le suivi des obstacles

Le pistage ou suivi des obstacles est utilisé afin d'extraire l'information d'un système dynamique. Pour définir le problème de suivi, introduisons le modèle d'évolution (vitesse constante) propre à notre application :

$$X_{k+1} = FX_k + V_k \quad (1)$$

où $X_k = (x, \dot{x}, y, \dot{y}, z, \dot{z})^t$ est le vecteur d'état, F la matrice de transition qui modélise l'évolution de X_k, et V_k le bruit sur l'état considéré comme étant l'accélération.

$$F = \begin{matrix} 1 & T & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & T & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & T \\ 0 & 0 & 0 & 0 & 0 & 1 \end{matrix}$$

L'objectif du suivi est d'estimer récursivement X_k à partir des mesures :

$$Z_k = HX_k + W_k$$

où $Z_k = (x, y, z)^t$ est le vecteur de mesure, et W_k le bruit sur la mesure.

Le problème de suivi d'un obstacle, du point de vue bayésien, peut être vu comme le calcul récursif de degrés de croyance en l'état X_k au temps k, en prenant en compte l'historique des mesures $Z_{1:k} = \{Z_1, ..., Z_k\}$. Il est donc nécessaire de construire la densité de probabilité a posteriori (notée *pdf* dans la suite) $p(X_k / Z_{1:k})$. On considère que la *pdf* initiale est connue et est donnée par $p(X_0 / Z_0) \quad p(X_0)$. Alors, par principe, la pdf $p(X_k / Z_{1:k})$ peut être obtenue récursivement en deux étapes. La première étape consiste en la prédiction de $p(X_k / Z_{1:k-1})$ à partir de $p(X_{k-1} / Z_{1:k-1})$ et d'une densité de transition $p(X_k / X_{k-1})$. La deuxième phase utilise la règle de Bayes afin de mettre à jour les $p(X_k / Z_k)$ en fonction des nouvelles mesures.

3.3.1 Initialisation des pistes

On assimile un obstacle à une piste si sa variation de vitesse donnée par la position du centre de la cible dans trois images consécutives est inférieure à un seuil.

3.3.2 Maintenance des pistes

3.3.2.1 Filtre de kalman

Les filtres de Kalman sont des méthodes récursives basées sur la représentation de la densité de probabilité de l'état (conditionné par les mesures) par une loi normale d'espérance l'estimation de l'état X_k et de variance la covariance sur cet état P_k.

$$p(X_k / Z_k) \sim \mathrm{N}(X_k, P_k)$$

Cette représentation n'est valable que si les bruits sur l'état, et de mesure sont indépendants et gaussiens. Selon cette hypothèse on a :

$$V_k \sim \mathrm{N}(0, Q_k)$$

$$W_k \sim \mathrm{N}(0, C_k)$$

où Q_k et C_k modélisent respectivement le bruit sur l'état et le bruit sur la mesure.
Sous ces hypothèses, l'utilisation des équations de Kalman permet une estimation optimale de l'état de notre obstacle.

3.3.2.2 Filtre à particules

Les filtres particulaires sont des méthodes séquentielles de Monte-Carlo, basées sur la représentation de la densité de probabilité (de l'état conditionné par les mesures) par des masses (ou particules), qui peuvent être appliquées à tout modèle d'état, et qui généralisent les méthodes traditionnelles du filtrage de Kalman . Ils permettent une estimation récursive optimale du vecteur d'état même si le modèle d'état est non linéaire et le bruit sur la mesure est non gaussien.

Le filtrage particulaire se base sur l'estimation d'une densité de probabilité a posteriori à partir d'un ensemble de particules pondérées.

Le filtrage utilisé dans notre système est un filtre SIR (Sampling Importance Resampling). C'est une méthode de Monte-Carlo qui peut être appliquée aux problèmes de filtrage bayésien récursif.

La fonction qui va permettre d'effectuer le filtrage est donnée par :

$$\left\{ x_k^i, w_k^i \right\}_{i=1}^{N_p} = SIR \left\{ x_{k-1}^i, w_{k-1}^i \right\}_{i=1}^{N_p}, z_k$$

où N_p est le nombre de particules utilisées, x_k^i l'état pour la particule i à l'instant k et w_k^i le poids associé à cette particule.

L'algorithme de cette fonction est décrit ci-dessous :

Pour
    Echantillonnage des :
    Calcul des poids :
FinPour
    Calcul du poids total :
Pour
    Normalisation :
FinPour
rééchantillonnage pour favoriser les particules de plus fort poids

Pour notre application, le filtrage se décompose donc suivant les étapes décrites ci-dessous :

1. Un jeu de particules est généré à partir de la valeur initiale du vecteur d'état X_0. Chaque particule a un poids associé égal à l'inverse du nombre de particules.
2. On applique à chaque particule le modèle d'évolution définie en (1). Une prédiction de l'état de chaque particule est alors calculée.
3. Le poids associé à chaque particule est alors calculé en fonction des nouvelles observations. Une normalisation de ces poids est effectuée.
4. On tire (avec remplacement) N particules parmi le jeu courant, de façon à favoriser les particules de plus forte vraisemblance (de plus fort poids). Le poids associé à chaque particule est de nouveau égal à l'inverse du nombre de particules.
5. L'espérance du jeu de particules fournit une estimation de l'état.
6. bouclage à l'étape 2.

3.3.3 Association de données

Dans la section précédente, nous avons présenté différents types de filtres de poursuite qui permettent une estimation optimale de l'état de l'obstacle à suivre. Ces filtres ont pour rôle de suivre une trajectoire étant données les observations qui lui sont associées. Cette association observation-état permet de sélectionner l'obstacle, si il existe, correspondant au mieux à la piste. Il existe de nombreux critères dans la littérature pour prendre la décision d'associer un obstacle à une piste. Ces critères dépendent essentiellement de la complexité de la tâche du pistage, comme par exemple le nombre de cibles, les manœuvres effectuées par les obstacles, le nombre de fausses alarmes. On pourra citer par exemple comme méthodes d'association celles à hypothèse unique (Plus Proche Voisin, Voisins des Voisins) ou à hypothèse multiple (PDAF, JPDAF,…) . Les méthodes à hypothèse unique sont adaptées aux environnements contenant peu d'événements [Hutb95]. On utilisera donc cette méthode puisque dans un contexte autoroutier le nombre d'obstacles visibles à l'avant du véhicule est réduit, les manœuvres effectuées par les obstacles sont très limitées car il n'y a pas de croisement. De plus, l'algorithme de détection d'obstacles par Lidar permet l'élimination de la plupart des fausses alarmes et délivre des mesures 3D précises (voir section 3.2). On utilisera donc une méthode d'association basée sur la recherche du plus proche voisin. Cette association est une étape du module de pistage décrit Figure 26. Pour chaque piste, il existe un module de ce type.

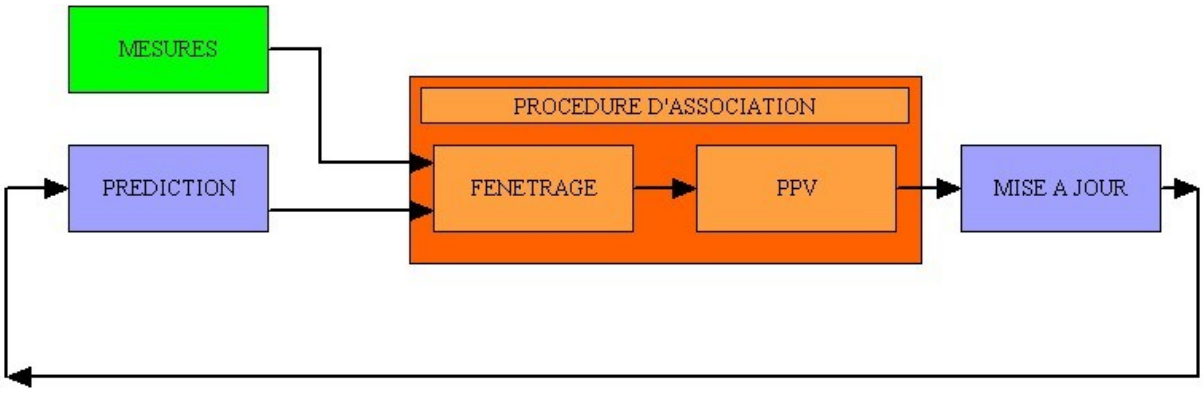


Figure 26. Module de pistage

Finalement, cette procédure d'association est réalisée en deux phases.Une opération de fenêtrage est réalisée afin de sélectionner les obstacles ayant la plus forte probabilité d'entretenir la piste et une étape d'association de données afin de sélectionner la mesure adéquate parmi toutes les mesures

présente dans la fenêtre de validation. Dans la plupart des cas rencontrés, une seule mesure appartient à la fenêtre de validation. La procédure de recherche du plus proche voisin est donc parfois inutile.
Le fenêtrage est utilisé afin de sélectionner les mesures qui ont la plus forte probabilité d'appartenir à la piste. Ceci évite de rechercher les mesures dans l'espace total des mesures et élimine donc les calculs inutiles pour l'association de données.

Soit $Inn = Z_k - HX_k$ l'innovation, différence entre mesure réelle et mesure prédite, et $S = HP_kH^t + C_k$ La covariance de l'innovation. La distance de Mahalanobis entre Z_k et HX_k est donnée par :

$$d = Inn^t S^{-1} Inn$$

En supposant que la mesure, de dimension m est gaussienne, la distribution de probabilité de la variable d suit une distribution du χ^2 à m degrés de liberté.
Finalement, le test pour savoir si Z_k et HX_k proviennent de la même distribution à $N\%$ prés est donnée par :

$$d \quad \gamma \qquad (1)$$

où γ est un paramètre issu des tables du χ^2 à m degrés de liberté avec un niveau de confiance de $N\%$. Ici on utilise, $N = 95\%$ et $m = 3$ donc $\gamma = 7,8$.
Enfin, si plusieurs obstacles tombent dans la fenêtre de validation définie par l'équation (1), on sélectionne l'obstacle ayant la plus petite valeur de distance d.

3.3.4 Mort des pistes

Lorsque les obstacles sortent du champ de vue du capteur ou sont occultés, l'algorithme utilise une approche basée sur l'âge de la piste : les paramètres de l'obstacle sont mis à jour par la prédiction de l'état, l'obstacle peut donc être associé de nouveau quand il réapparaît. La piste est dite morte si elle ne réapparaît pas après un temps fixé (proche de une seconde).

3.3.5 Résultats

Les premiers résultats présentent une séquence de suivi par filtrage de Kalman et par filtrage particulaire dans un cas mono cible.
Les figures représentent respectivement la position latérale (Figure 27), la position longitudinale (Figure 28) et la vitesse radiale (Figure 29) pour les deux types de filtres. Les mesures de position issues du traitement des données laser apparaissent en noir. La Figure 30 permet de repérer cinq instants de la séquence (a,b,c,d,e).

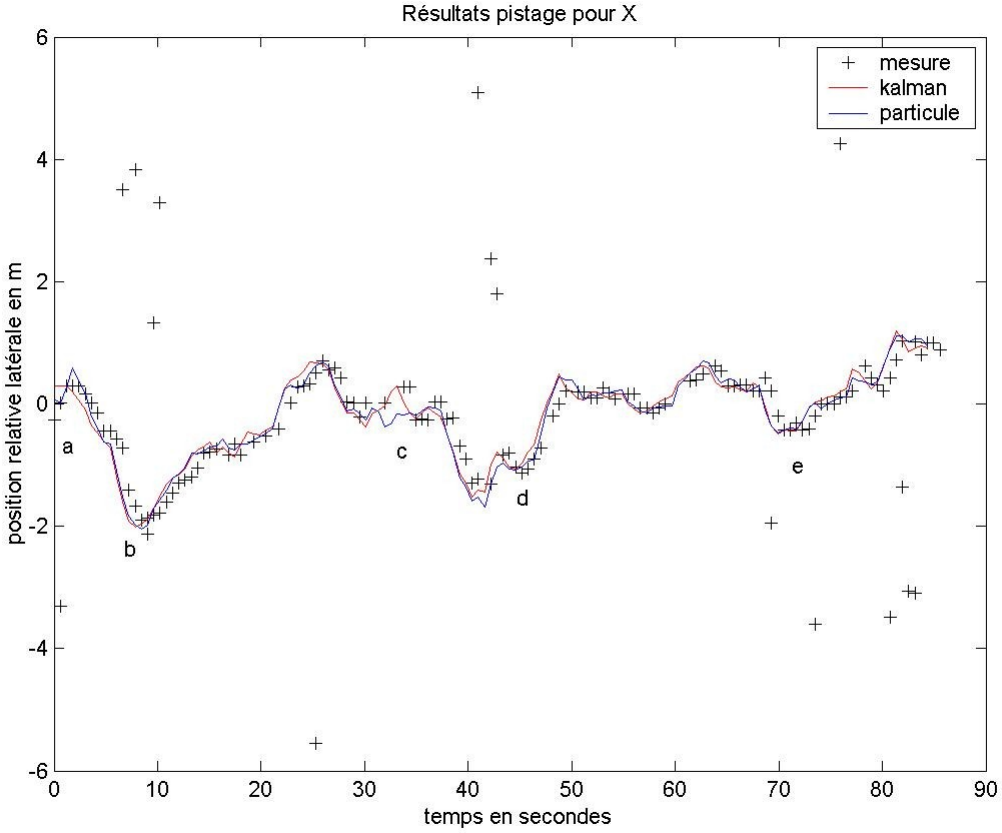


Figure 27. Résultats pistage pour X dans un cas monocible

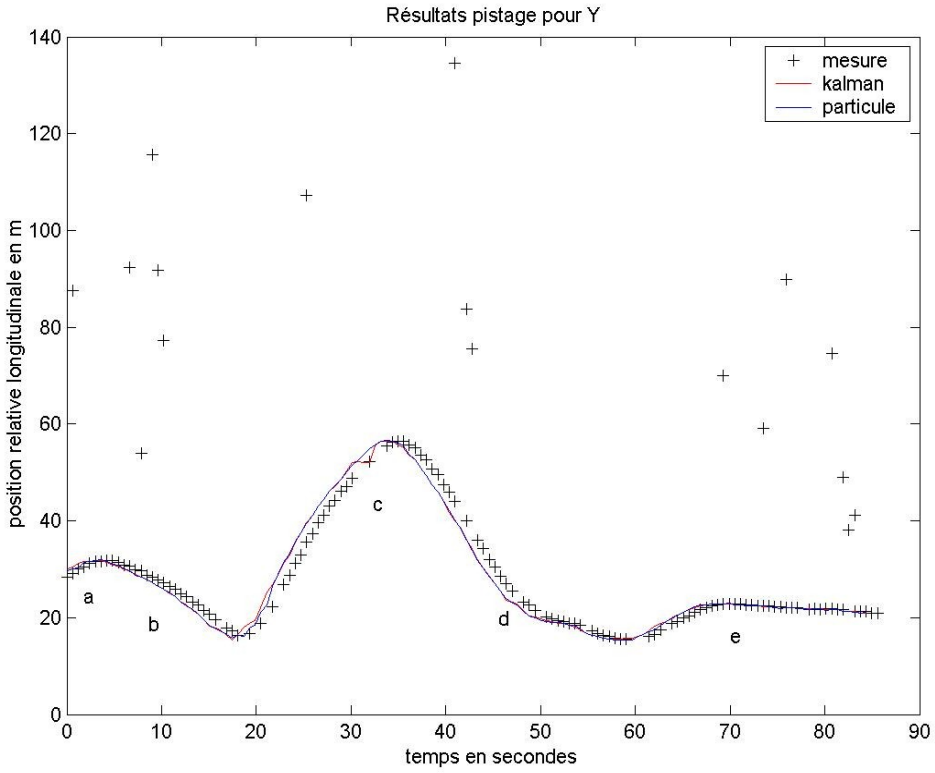


Figure 28. Résultats pistage pour Y dans un cas monocible

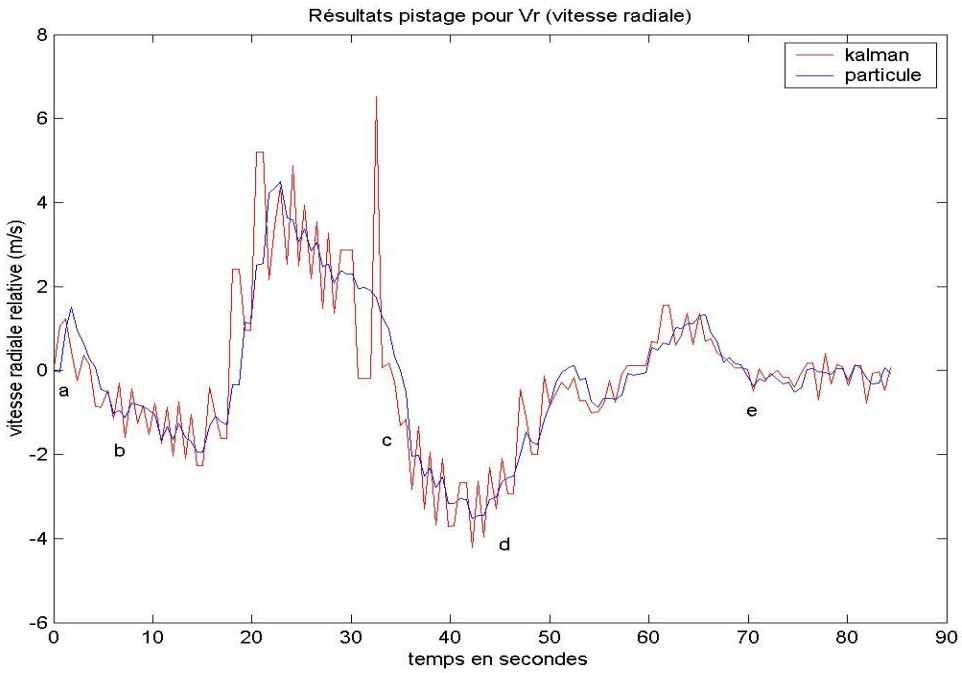


Figure 29. Résultats pistage pour Vr dans un cas monocible

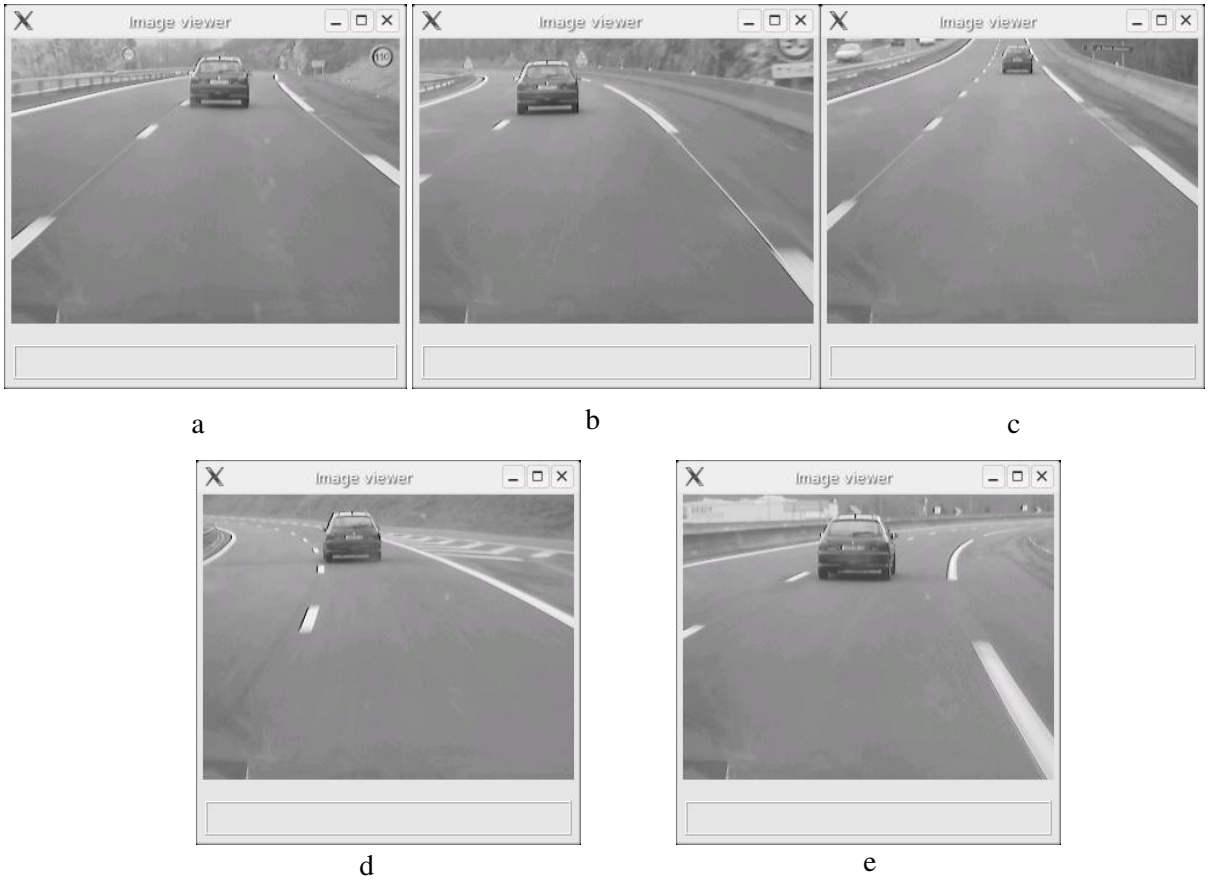


Figure 30. Instants de la séquence monocible

Au vu de ces résultats, les deux filtres donnent quasiment les mêmes estimées en position (voir Figure 27 et Figure 28). En effet, le modèle étant linéaire et les bruits gaussiens, l'utilisation du filtre de Kalman est optimale pour les estimées en position. L'intérêt de l'utilisation du filtre à particule est souligné par les résultats sur l'estimation de la vitesse radiale. On remarque que le filtrage particulaire permet une estimation de cette vitesse plus lissée que l'estimation par filtrage de Kalman (voir Figure 29).

Pour les résultats suivants, dans un cas multicible, on ne présente que les estimées du filtre à particule. Dans les figures ci-dessous, les estimées en position et en vitesse des différentes pistes sont représentées avec des couleurs différentes (Figure 31, Figure 32, Figure 33). La Figure 34 permet de repérer plusieurs instants de la séquence.

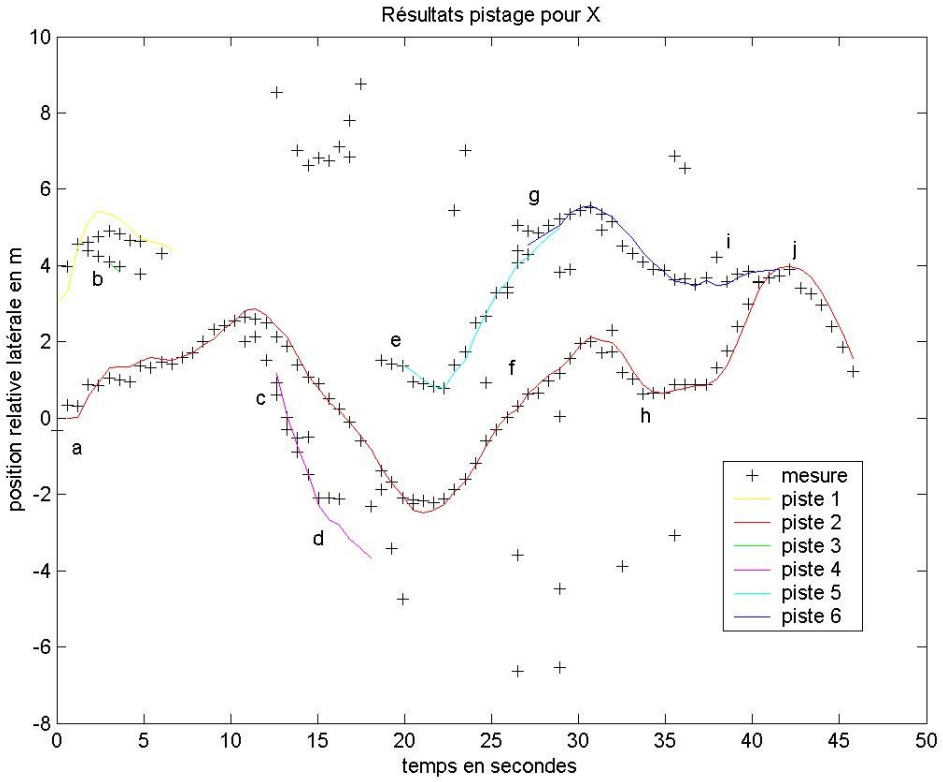


Figure 31. Résultats pistage pour X dans un cas multicible

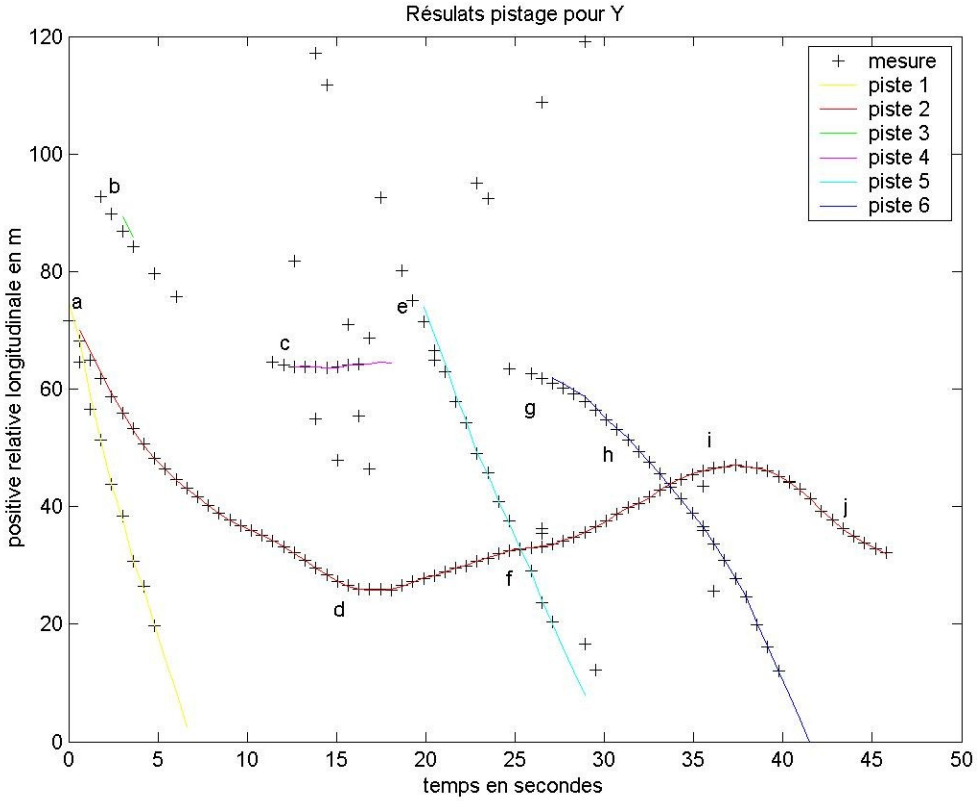


Figure 32. Résultats pistage pour Y dans un cas multicible

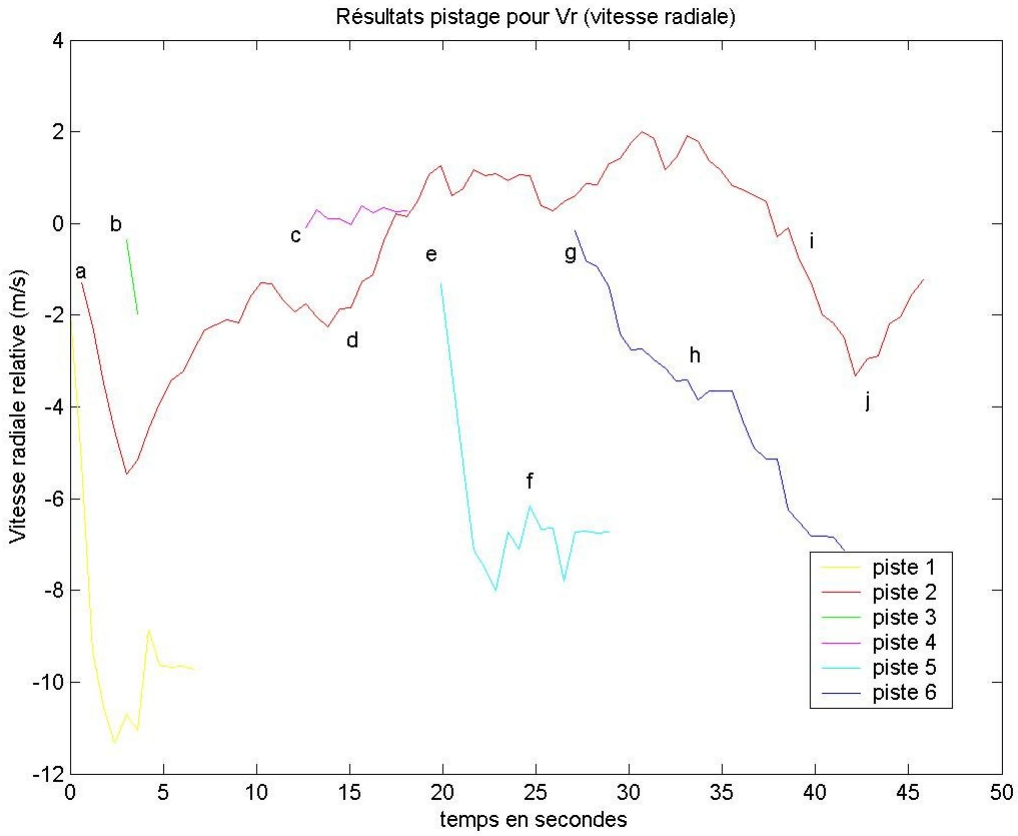


Figure 33. Résultats pistage pour Vr dans un cas multicible

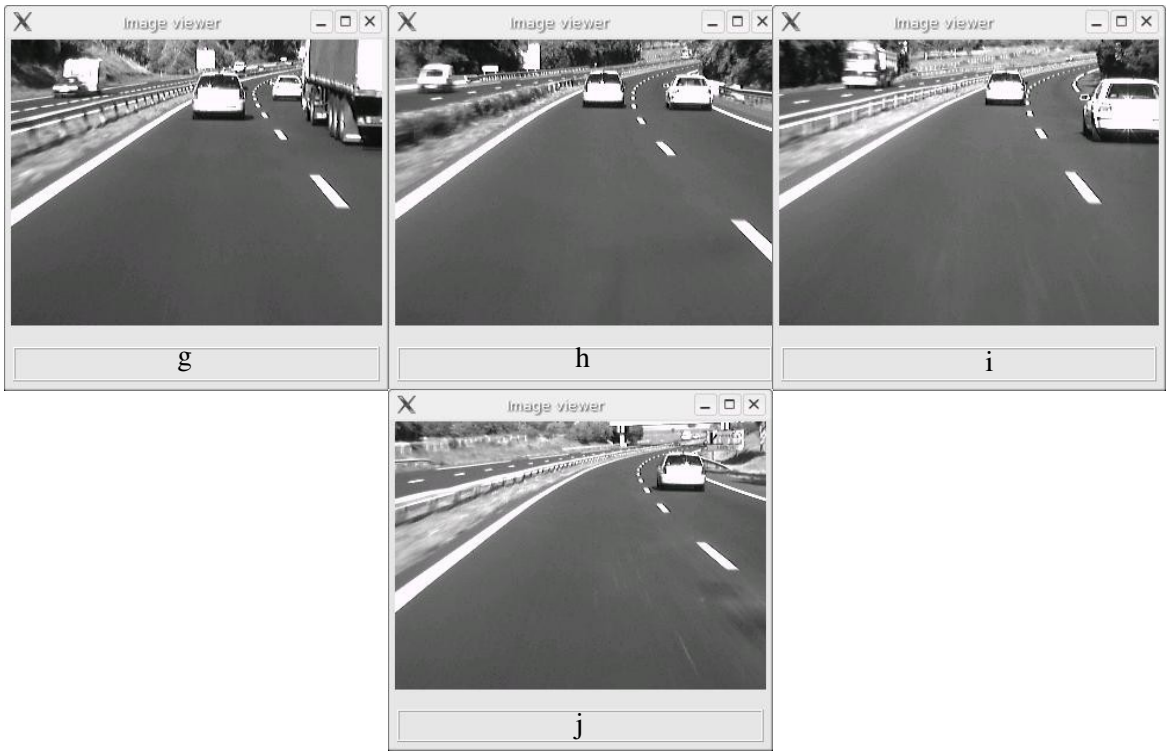


Figure 34. Instant de la séquence multicible

Ces résultats montrent la capacité du système de pistage multicible dans un contexte autoroutier. On remarque que l'on est capable de détecter et pister plusieurs types d'obstacles (voitures et camions). De plus, aucune fausse piste n'apparaît.

3.4 Conclusion

Les résultats obtenus par la technologie laser pour la détection d'obstacles en général sont satisfaisants. Ils permettent d'obtenir un bon compromis entre le niveau de reconnaissance, la qualité de la localisation et la cinématique qui en est déduite. Ces résultats sont confortés par l'évaluation effectuée par le thème 1 sur les différents systèmes de perception. Cependant, la configuration actuelle du système de déflection ne permet pas d'obtenir une cadence de rafraîchissement des données suffisante. Cette conclusion ne remet pas en cause la pertinence de ce type de capteurs pour de telles applications. En revanche, elle met en exergue la nécessité d'accélérer les recherches autour de la déflection du faisceau laser.

La méthode proposée constitue la base des solutions implantées dans les fonctions « anticollision » et « gestion des interdistances » où elle est déclinée plus spécifiquement pour répondre aux deux cahiers des charges.

3.5 Bibliographie

[AMC+00] R. Aufrère, F. Marmoiton, R. Chapuis, F. Collange, and J. P. Dérutin, « Détection de route et suivi de véhicules », Traitement du Signal, vol. 17 : 233-247, 2000.

[BarS88] Y. Bar-Shalom, T.E. Fortmann, «Tracking and data association», Mathematics in Science and Engineering. Academic Press, 1988.

[Chec 96] P. Checchin. «Segmentation d'images de profondeur», Thèse, Université Blaise Pascal de Clermont-Ferrand, France, 1996.

[Cla03] Xavier Clady, « Contributions à la navigation autonome d'un véhicule automobile par vision », Thèse présentée à l'Université Blaise Pascal Clermont-Ferrand II, 2003.

[DWM94] D.Koller, J. Weber, and J. Malik., « Robust Multiple Car Tracking with Occlusions Reasoning », Proceedings of Third European Conference on Computer Vision, Stockholm, Sweden, May 2-6, 1994., pages. 189-196.

[Hutb95] D. Hutber, «Suivi multi-capteurs de cibles multiples en vision par ordinateur, appliqué à un véhicule dans un environnement routier», Thèse, Université de Nice – Sophia Antipolis, France, 1995.

[Joua99] S. Jouannin, «Association et fusion de données : application au suivi et à la localisation d'obstacles par radar à bord d'un véhicule intelligent», Thèse, Université Blaise Pascal de Clermont-Ferrand, France, 1999.

[OFG97] E. Osuna, R. Freund, and F. Girosi, « Support vector machines: Training and applications. », Artificial Intelligence Memo 1602, MIT A.I. Lab., 1997.

[TAG94] L.Trassoudaine, J. Alizon, and J. Gallice, « Road obstacle detection and tracking byanactive and intelligent sensing strategy », Machine Vision Applications, (7):165--177, 1994.

[Trass93] L. Trassoudaine, « Solution multisensorielle temps réel pour la détection d'obstacles sur route », Thèse, Université Blaise Pascal de Clermont-Ferrand, France, 1993.

4 Fusion de données laser 3D/vidéo dédiée à la fonction « Gérer les interdistances »

4.1 Introduction

L'étude présentée dans ce chapitre résulte des conclusions émanant du rapport R10 sur la détection d'obstacles par perception monocapteur. Il apparaît notamment que la télémétrie laser 3D permet d'obtenir d'excellents résultats en terme de reconnaissance et de localisation mais que l'état actuel de la technologie ne permet pas d'accéder à des cadences de détection suffisantes.

L'objet de la combinaison d'information est d'intégrer des mesures proprioceptives et extéroceptives afin de déterminer la voie de roulement de la voiture suivie. L'autre niveau de combinaison a pour but de palier la faible cadence de rafraîchissement des mesures laser en introduisant un nouveau process de vision pour maintenir le suivi des véhicules entre deux balayages laser. Ces développements spécifiques répondent aux besoins de la fonction ARCOS « gérer les interdistances ».

La première section présente l'étude de faisabilité et l'implémentation des capteurs proprioceptifs (gyromètre et angle volant) afin de faire un calcul de reconstruction et de prédiction de la route en complément du positionnement latéral par vision monoculaire. La deuxième section est consacrée à un rapide rappel sur la détection et à la localisation des obstacles par télémètre LASER. Enfin, la dernière section présente comment intégrer le suivi par vision dans le processus global de poursuite du véhicule.

4.2 Reconstruction de la route à l'avant du véhicule expérimental

Les capteurs proprioceptifs sont les capteurs qui permettent de connaître l'état interne du véhicule. Ils sont ici utilisés pour en déduire l'allure de la route à l'avant du véhicule en prenant pour hypothèse qu'elle varie peu. Les capteurs que nous avons considérés pour cette étude sont le gyromètre et le capteur d'angle au volant.

4.2.1 Les capteurs proprioceptifs

4.2.1.1 Le gyromètre

Le gyroscope piézo-électrique est un capteur utilisé pour déterminer la vitesse angulaire d'un solide. En positionnant ce capteur sur le solide considéré, la vitesse angulaire peut être mesurée facilement même pour un solide se déplaçant avec son centre de gravité en mouvement continu.

Le principe de mesure de la vitesse angulaire consiste à déterminer la force de Coriolis mesurée par trois céramiques piézo-électriques. Le fonctionnement est le suivant : Les trois céramiques identiques de type piézo-électrique sont disposées sur les faces d'un élément vibrant de forme prismatique qui est excité en vibration par un oscillateur asservi en amplitude et en fréquence utilisant l'un des éléments piézo-électriques(1).
Les deux autres (2 et 3) détectent une modification du mode de vibration dont la cause est la vitesse angulaire.

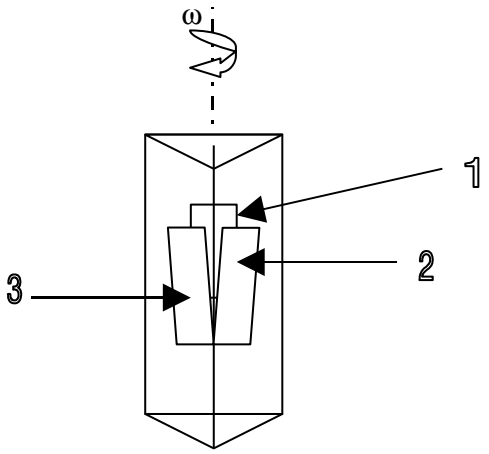


Figure 35 - Représentation du principe du gyromètre.

La vitesse angulaire est calculée par le gyromètre en soustrayant les amplitudes des signaux de sortie droite et gauche, qui sont données par les capteurs de détection droite et gauche. A la sortie du gyromètre, on trouve un signal analogique qui est directement proportionnel à la grandeur physique mesurée.
Le gyromètre « Gyrostar ENV05F-03 » a été choisi. Ces caractéristiques importantes sont reprises dans le tableau suivant.

xpdf: /home/ahmad/Desktop/GYRO/GYROSCOPE_SPECS.pdf

Type ENV-05F-03

Characteristic	Symbol	Condition	MIN.	STD.	MAX.	Unit
Supply voltage	Vcc		+4.5	+5.0	+5.5	VDC
Current consumption	Icc	at Vcc=5.0VDC	-	-	15	mA
Max. angular velocity	Omax		-1/3 PI (-60)	-	+1/3 PI (+60)	rad/s (deg/s)
Output	Vo	angular velocity = 0 at -30~80℃	2.150	2.500	2.850	VDC
Scale factor	Sv	at -10~60℃ at -30~80℃	23.0 21.7	25.0 25.0	27.0 28.3	mV/deg/s
Asymmetry CW & CCW			-	-	3	deg/s
Temp. coefficient Scale factor		reference : Ta at -10~60℃ at -30~80℃	- -	- -	±5 ±10	%FS
Drift		at -30~80℃	-	-	9	deg/s
Start up		Measure Vo after 5s	-	-	±1	deg/s/10mi
Noise level		10kHz noise	-	-	20	mVrms
Linearity		in the Omax	-	-	0.5	%FS
Response		Phase delay : 90deg	-	10	-	Hz
Dependence on Supply voltage Output			0.8	-	1.2	
Scale factor			0.8	-	1.2	
Operating temp. range	Topr		-30	-	80	℃
Storage temp. range	Tstg		-40	-	85	℃
Weight			-	-	20	g
Dimension			11.5(D) x 19.6(W) x 23.2(H) mm			

Page 3 of 10 +2 Quit

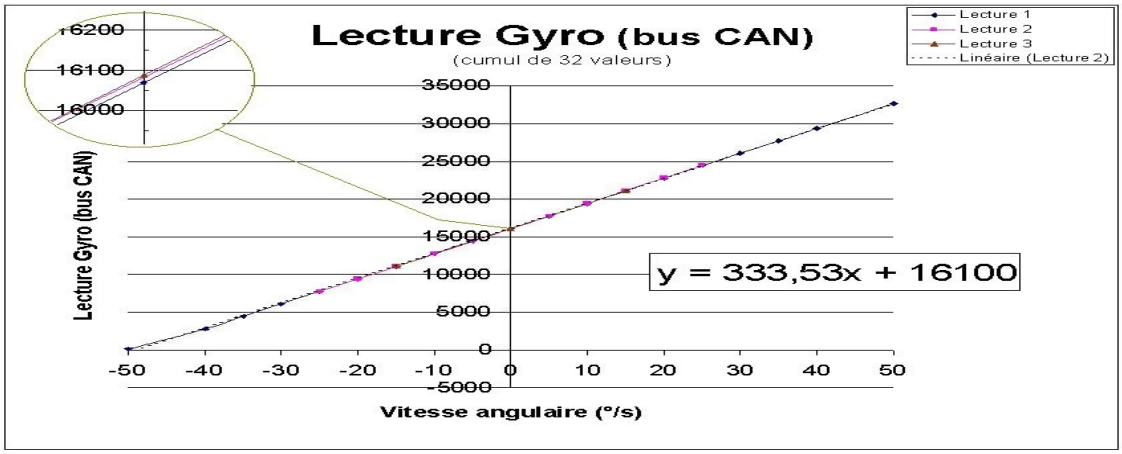
Tableau 5 -Caractéristiques importantes du gyromètre (toutes les observations sont obtenues à 25°c)

Pour mettre en forme le signal, le système global à été conçu de telle manière que la grandeur à mesurer ne dépasse jamais +/- 50°/sec.

- Pour une variation de la vitesse angulaire de 100°/s on obtient une variation de 2,5V :
  100° /s x 0.025V/°/sec =2,5V
- Pour 0°/s on obtient 2,5V :
  La tension de sortie minimale : 2.5V - (50°/s x 0.025V/°/s) = 1,25V
  La tension de sortie maximale : 2.5V + (50°/s x 0.025V/°/s) = 3,75V
- Notre mise en forme : S = G x (sortie capteur + offset)
  Le gain pour une plage de 5V : G = 5V/2,5V = 2V
  L'offset pour 0v à –50°/s : Offset = -1,25V

Le gyroscope a été calibré en l'installant sur la main d'un robot manipulateur de haute précision « Motoman ». Les trajectoires réalisées sont des cercles parcourus à des vitesses angulaires s'étalant de 15°/sec à 45°/sec. Les différentes valeurs ont été récupérées via le CAN. En utilisant une plage de 32 valeurs différentes, on obtient :

Y=333.53x + 16100
i.e. ω=(32*(lecture CAN))/n*333)-48.3 °/sec où n= la taille d'échelle



ensemble piCan3 posé sur le robot motoman commandé en vitesse
nœud piCan3 configuré pour envoyer le cumul de 32 mesures à 3,2ms soit 102,4ms entre chaque trame
Lecture depuis le bus CAN sur un PC
moyenne de 5 à 20 lectures suivant la vitesse (5_rapide, 20 lent)
signe positif sens horaire
les lectures semblent avoir augmentées au 2e passage (offset temperature ?)

Figure 36 - Graphe de calibrage du gyromètre.

Les données brutes ainsi obtenues sont ensuite filtrées par un filtre passe bas afin de déduire un rayon de courbure lissé (Figure 37 et Figure 38).

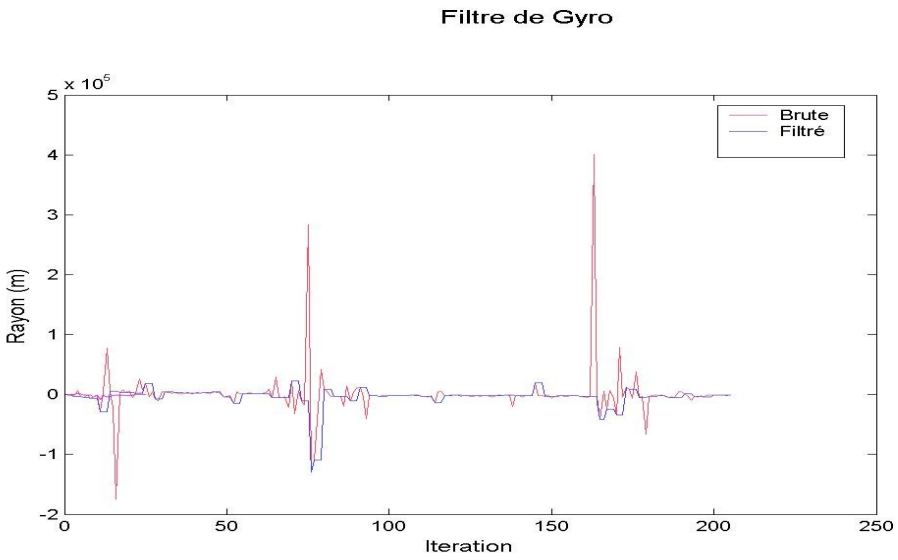


Figure 37 - Rayon filtré à vitesse élevée.

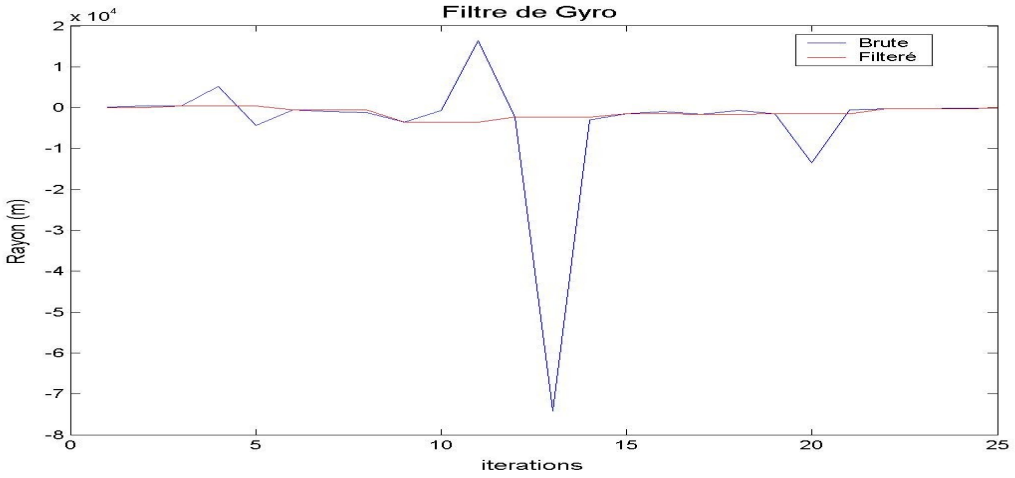


Figure 38 - Rayon filtré à basse vitesse.

4.2.1.2 Capteur de l'angle au volant

La mesure du rayon de courbure de la chaussée a également été envisagée en exploitant la mesure de l'angle au volant. Le capteur de l'angle au volant consiste en un potentiomètre fixé sur le volant qui est une partie d'un pont diviseur. Ainsi quand le volant tourne, il y a un changement dans la résistance à travers le potentiomètre et donc une variation de tension dans le circuit. Cette variation de tension est

utilisée pour mesurer l'angle de rotation du volant ou « l'angle volant ». Il permet de déterminer l'angle des roues que nous utiliserons dans le modèle cinématique du véhicule.

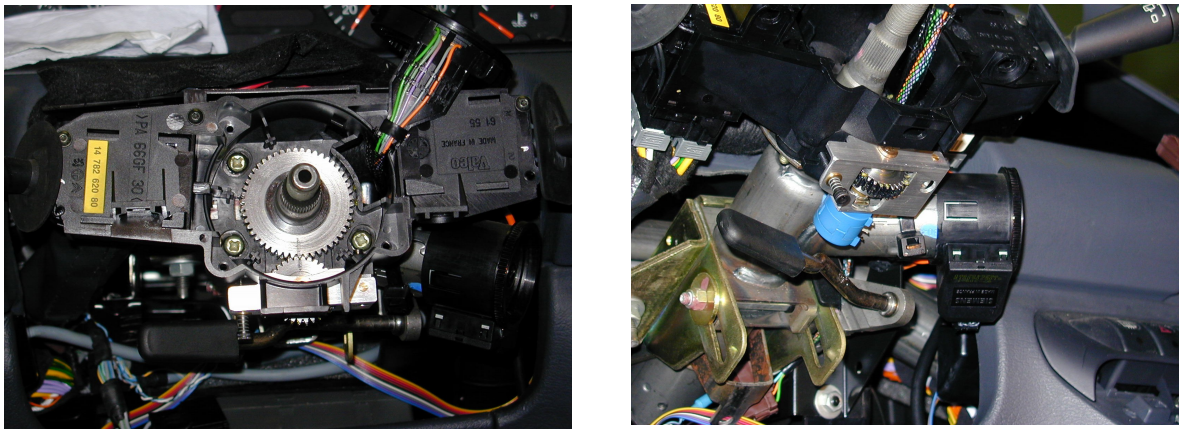


Figure 39 - Capteur de l'angle au volant.

Pour calibrer le capteur, la roue a été placée sur un compas et elle a été tournée dans les deux directions. L'angle a été mesuré avec le compas conjointement à la donnée mesurée par le capteur de l'angle volant. Les différentes valeurs ont été récupérées via le CAN. La formule de calibrage utilise l'équation caractéristique d'une droite. En appliquant une régression linéaire, le capteur est étalonné avec la relation suivante :

$Y = 0.0198x - 10.75$ où x est la lecture CAN et Y l'angle des roues.

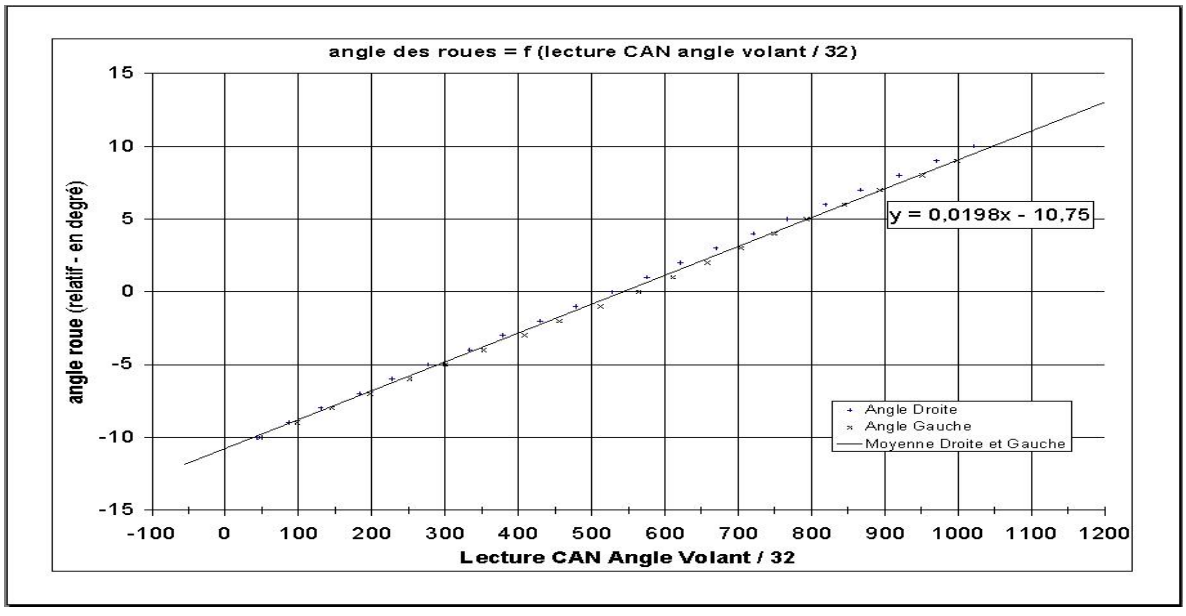


Figure 40 - calibrage de l'angle de roue

4.2.2 Localisation et reconstruction de la chaussée à l'aide de données proprioceptives

4.2.2.1 Localisation

Pour déterminer les capacités des capteurs et vérifier leurs équations de calibrage, on a commencé avec une localisation (observation à chaque instant) utilisant les capteurs. Le capteur gyromètre et l'angle volant ont été calibrés séparément. Ils étaient déjà montés sur VELAC qui été utilisé pour les expérimentations. Tout d'abord, pour le test on a choisi le boulevard Bingen (2 km) ainsi que l'autoroute A-75 (40 km) et quelques rond-points spécifiques car on dispose d'un modèle numérique ce qui constitue une vérité de terrain. On a utilisé le modèle bicyclette pour la représentation de l'évolution de VELAC.

L'évolution des variables d'états est représentée par le système suivant :

Gyromètre	Angle volant
$X_{k+1}=X_k + v*t*sin(\theta_k)$	$X_{k+1}= X_k + v*t*sin(\theta_k)$
$Y_{k+1} = Y_k + v*t*cos(\theta_k)$	$Y_{k+1} = Y_k + v*t*cos(\theta_k)$
$\theta_{k+1}= \omega*t + \theta_k$	$\theta_{k+1}= \theta_k + tan(\phi)/l$

où ω est la vitesse angulaire mesure par le gyromètre, v la vitesse linaire déduite des mesures odométriques, t la période d'échantillonnage, ϕ l'angle de roues, $\theta_{k\,l}$ l'angle de cap du véhicule, et l la distance entre les deux axes.

Les premiers tests portent sur une évaluation des résultats que l'on pourrait obtenir en termes de localisation à l'aide de ces seules informations proprioceptives. L'objectif fixé pour le projet porte sur la reconstruction de la chaussée à chaque instant est moins contraignant puisque dans ce cas, les erreurs ne s'accumulent pas. Il est cependant intéressant d'avoir une idée des résultats envisageables dans une application plus complexes telle que la localisation.
Les résultats obtenus sont corrects avec le gyromètre mais l'erreur est plus importante avec le capteur de l'angle volant. L'erreur augmente évidemment avec la distance parcouru (voir Figure 41 et Figure 42). Sur la Figure 41 nous pouvons voir que les graphiques aller et retour (trace de localisation) obtenus par le gyromètre sont parallèles ainsi nous pouvons dire que l'équation de calibrage pour le gyromètre est correcte. Ceci n'est pas vrai pour le capteur de l'angle au volant. On voit que la localisation faite par le gyroscope est presque exacte jusqu'à une certaine distance. Dans le cas du boulevard Bingen la divergence commence à peu près à 0,85 km et pour l'autoroute A-75 à peu près à 19 km. Les observations qualitatives faites par le gyromètre sont donc plus précises que celles faites avec l'angle au volant.

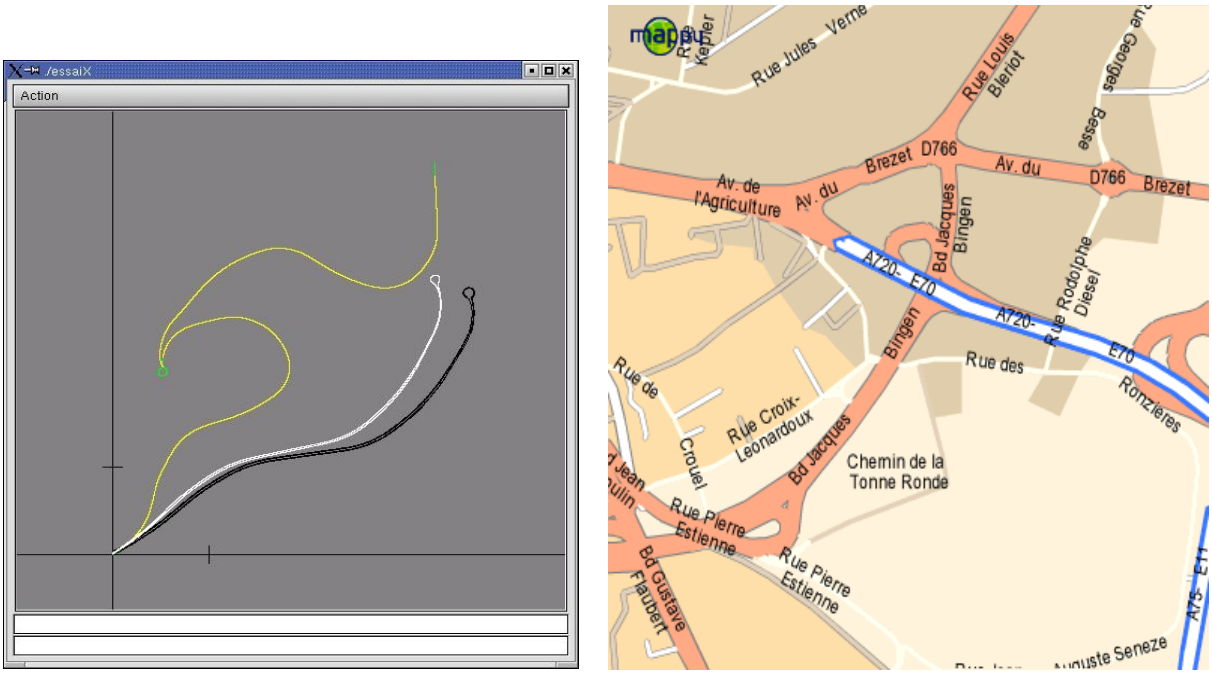


Figure 41 - Boulevard Bingen (noir = GPS, blanche = gyromètre, jaune = angle volant)

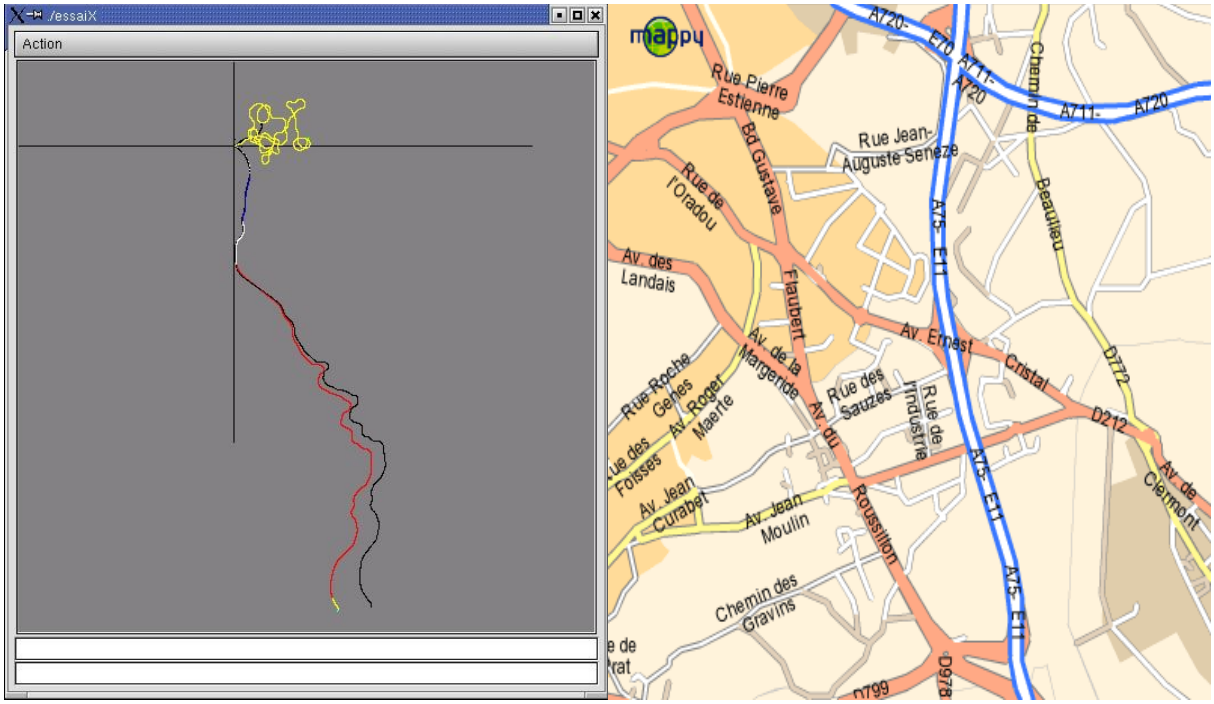


Figure 42 - Autoroute A-75 (noir = GPS, rouge = gyromètre, jaune = angle volant)

4.2.3 Reconstruction de la chaussée

Pour prédire l'allure de la route avec le gyromètre et l'angle des roues on utilise les algorithmes décrits dans les sections suivantes :

4.2.3.1 Prédiction avec le gyromètre

Pour faire la prédiction de la route avec le gyromètre on utilise les hypothèses et des notations suivantes :

- Largeur de la voie = 2a (cette supposition est vraie comme la largeur d'une autoroute est normalement la même partout ≈ 3,5m)
- Chaque segment de construction de la route = *y
- Vitesse constante = v
- Vitesse angulaire = ω
- Chaque courbe est une portion de cercle (cette supposition est vraie pour l'autoroute car il n'y a pas de courbes très fortes mais pour les routes comme en centre ville on peut faire la supposition que chaque courbe est un segment de la courbe tangente).

On suppose que la voiture se déplace le long du centre de la voie de la route avec une vitesse v et les conditions initiales : $X_0=0$, $Y_0=0$, $Xl_0=0, Yl_0=0, Xr_0=0, Yr_0=0$ (dans le repère du véhicule)

Si elle se déplace sur la route droite ; ω = 0, on peut faire la prédiction de la route devant par (Xr : position bord gauche, Xl : position bord droit, Xc position ligne centrale) :

Centre	bord droit	bord gauche
$X_1 = X_0$	$Xr_1 = Xr_0 + a/2$	$Xl_1 = Xl_0 - a/2$
$Y_1 = Y_0 + *y$	$Yr_1 = Yr_0 + *y$	$Yl_1 = Yl_0 + *y$
$X_2 = X_1$	$Xr_2 = Xr_1 + a/2$	$Xl_2 = Xl_1 - a/2$
$Y_2 = Y1 + *y$	$Yr_2 = Yr_1 + *y$	$Yl_2 = Yl_1 + *y$
----------	----------	----------
----------	----------	----------
$X_n = X_{n-1}$	$Xr_n = Xr_{n-1} + a/2$	$Xl_n = Xl_{n-1} - a/2$
$Y_n = Y_{n-1} + y*$	$Yr_n = Yr_{n-1} + *y$	$Yl_n = Yl_{n-1} + *y$

Si elle se déplace sur une route courbe avec une vitesse v ; ω ≠ 0 (b) on trouve la vitesse v grâce à l'odomètre et la vitesse angulaire ω grâce au gyromètre. Le rayon de ce cercle vaut $r = v/\omega$.

On suppose que les cordonnées du centre de ce cercle sont *(r, 0)*. L'équation de ce cercle est donc :

$(x - r)^2 + y^2 = r^2$
$x^2 - 2xr + y^2 = r^2$
$x1 = r + sqrt(r^2 - y^2)$ *ou* $x2 = r - sqrt(r^2 - y^2)$

On suppose que ce segment est seulement une portion du cercle, on choisit:

$x = min(|x1|, |x2|)$

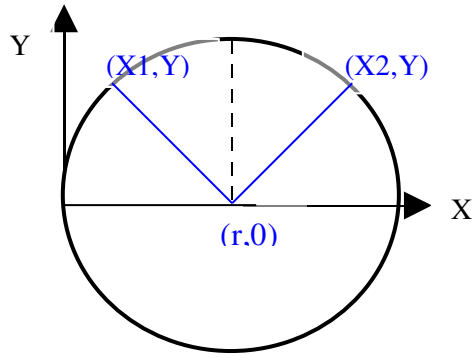


Figure 43 - Représentation de la route prédite.

Ainsi les bords de voie sont obtenus par :

Centre	bord droit	Chausse gauche
$Y_1 = Y_0 + *y$	$Yr_1 = ((r - a/2\ ).Y_1)/ r$	$Yl_1 = ((a/2+r\ ).Y_1) / r$
$X_1 = r - sqrt(r^2 - (Y_1)^2) + X_0$	$Xr_1 = r - (\ r + a/2\ ).(r - X_1)/r$	$Xl_1 = r - (\ r - a/2\ ).(r - X_1)/r$
$Y_2 = Y_1 + *y$	$Yr_2 = ((r - a/2\ ).Y_2)/ r$	$Yl_2 = ((a/2+r\ ).Y_2) / r$
$X_2 = r - sqrt(r^2 - (Y_2)^2) + X_1$	$Xr_2 = r - (\ r + a/2\ ).(r - X_2)/r$	$Xl_2 = r - (\ r - a/2\ ).(r - X_2)/r$
----------	----------	----------
----------	----------	----------
$Y_n = Y_{n-1} + *y$	$Yr_n = ((r - a/2\ ).Y_n) / r$	$Yl_n = ((a/2+r\ ).Y_n) / r$
$X_n = r - sqrt(r^2 - (Y_n)^2) + X_{n-1}$	$Xr_n = r - (\ r + a/2\ ).(r - X_n)/r$	$Xl_n = r - (\ r - a/2\ ).(r - X_n)/r$

Si la voiture suit une courbe dans l'autre direction, ω est négative donc la valeur de rayon r va devenir négative ($r = v/\omega$) i.e. le centre du cercle est en $(-r, 0)$. On utilise donc les équations avec r négatif :

$Xn = r + sqrt(r^2 - (Yn)^2) + Xn\text{-}1$ $Xrn = r - (\ r + a/2\ ).(r - Xn)/r$ $Xln = r - (\ r - a/2\ ).(r - Xn)/r$

4.2.3.2 Prédiction avec le capteur de l'angle au volant

Pour obtenir la prédiction de l'allure de la route avec l'angle volant nous utilisons les mêmes hypothèses. Nous utilisons également le modèle bicyclette pour l'évolution du véhicule VELAC.

Si le véhicule se déplace sur une voie rectiligne; $\omega = 0$ et $\phi = 0$ (a) sinon $\omega \neq 0$ et $\phi \neq 0$

On peut faire la prédiction des bords de la route par :

Centre	Bord droit	Bord gauche
$Xc_{k+1}=X_k + v*t*\sin(\theta_k)$	$Xl_{k+1}=X_k + v*t*\sin(\theta_k)$	$Xr_{k+1}=X_k+v*t*\sin(\theta_k)$
$Y_{k+1}=Y_k + v*t*\cos(\theta_k)$	$Y_{k+1}=Y_k + v*t*\cos(\theta_k)$	$Y_{k+1}=Y_k+v*t*\cos(\theta_k)$
$\theta_{k+1}=\theta_k + \tan(\phi)/l$	$\theta_{k+1}=\theta_k + \tan(\phi)/l$	$\theta_{k+1}=\theta_k + \tan(\theta)/l$
$Xc_{k+2}=X_{k+1}+v*t*\sin(\theta_{k+1})$	$Xl_{k+2}=X_{k+1} + v*t*\sin(\theta_{k+1})$	$Xr_{k+2}=X_{k+1} + v*t*\sin(\theta_{k+1})$
$Y_{k+2}=Y_{k+1} + v*t*\cos(\theta_{k+1})$	$Y_{k+2}=Y_{k+1} + v*t*\cos(\theta_{k+1})$	$Y_{k+2}=Y_{k+1} +*t*\cos(\theta_{k+1})$
$\theta_{k+2}=\theta_{k+1} + \tan(\phi)/l$	$\theta_{k+2}=\theta_{k+1} + \tan(\phi)/l$	$\theta_{k+2}=\theta_{k+1} +\tan(\phi)/l$
----------	----------	----------
----------	----------	----------
----------	----------	----------

et on généralise par

$Xc_n=X_{n-1} + v*t*\sin(\theta_{n-1})$	$Xl_n=X_{n-1} + v*t*\sin(\alpha_{n-1})$	$Xr_n=X_{n-1} + v*t*\sin(\alpha_{n-1})$
$Y_n=Y_{n-1} + v*t*\cos(\alpha_{n-1})$	$Y_n=Y_{n-1} + v*t*\cos(\alpha_{n-1})$	$Y_n=Y_{n-1} + v*t*\cos(\alpha_{n-1})$
$\theta_n=\alpha_{n-1} + \tan(\phi)/l$	$\theta_n=\theta_{n-1} + \tan(\phi)/l$	$\theta_n=\theta_{n-1} + \tan(\phi)/l$

Où Y_n est la distance longitudinale maximale dans le repère Velac pour laquelle on fait la prédiction, l est la distance entre les deux axes, *y ou v*t est le déplacement parcouru par le véhicule (VELAC) mesuré par l'odomètre.

4.2.3.3 Introduction de la position latérale du véhicule sur la chaussée

La localisation du véhicule expérimental sur la route est très importante comme la prédiction de la route à l'avant est basée sur la position actuelle. Pour l'étape de prédiction nous avions fait la supposition selon laquelle notre voiture devait être au milieu de la voie. Un développement identique peut être appliqué en intégrant la position latérale du véhicule sur la chaussée. Cela permet donc de reconstruire la route avec précision et donc de juger si les obstacles sont sur cette route prédite, ou non ; et enfin de déterminer s'ils sont une menace potentielle.

Figure 44 - Localisation de la chaussée.

4.3 Détection des obstacles sur la route

Après l'étape de prédiction, l'étape suivante consiste à détecter les véhicules. Nous utilisons un télémètre laser à balayage 3D. Ce capteur est monté sur Velac. De plus amples détails sont donnés dans le rapport R9 du thème 1.

4.3.1 Description du capteur

Le capteur utilisé est un « 3D-Laser Mirror Scaner LMS-Z210-60 » de la société Riegl. Le mécanisme de balayage du télémètre consiste en un miroir polygonal tournant à quatre faces. L'angle de balayage est de 60° en horizontal et de 4.3° vertical. Le principe est basé sur la mesure du temps de vol des impulsions laser infrarouges. La grande sensibilité du télémètre lui permet de détecter la lumière diffusée.

Figure 45 - 3D-Laser Mirror Scaner ' LMS-Z210-60' dans le pare-chocs de VELAC.

La caractéristique principale de ce télémètre est sa portée de 100 mètres et sa faible divergence.

4.3.2 Algorithme de détection

Pour la détection des obstacles nous utilisons les deux parties de l'algorithme de détection. La première partie concerne la segmentation de l'image 3D et la deuxième la reconnaissance de la forme de l'obstacle.

Figure 46 - Détection d'obstacle avec le télémètre laser. Figure (a) montre l'image 3d de télémètre avec l'obstacle et figure(b) montre l'image 2d avec l'obstacle.

Une fois les différentes régions de l'image 3D déterminées, leurs dimensions sont comparées avec les différents modèles d'obstacles (voiture, camion, …). On en déduit ensuite le centre des obstacles détectés.

4.3.3 Le positionnement des obstacles

Pour déterminer si l'obstacle est sur la route ou non, il s'agit de faire collaborer les deux algorithmes (prédiction de l'allure de la voie et détection de véhicule).

Grâce à l'algorithme de détection nous pouvons trouver la position des véhicules dans le repère VELAC ; profondeur (y_{obs}) et abscisse (x_{obs}). A cette distance longitudinale (y_{obs}), nous déterminons la position de la route, dans le même repère (x_l et x_r) grâce aux algorithmes de prédiction et déterminons ainsi si chaque obstacle détecté est situé sur la route ou non.

si l'obstacle est sur la route :

$x_l \leq x_{obs} \leq x_r$

Où $x_l, x_r = f(y_{obs})$: *pour l'angle volant* $Y[n] = y_{obs}$ *et pour le gyro* $y = y_{obs}$

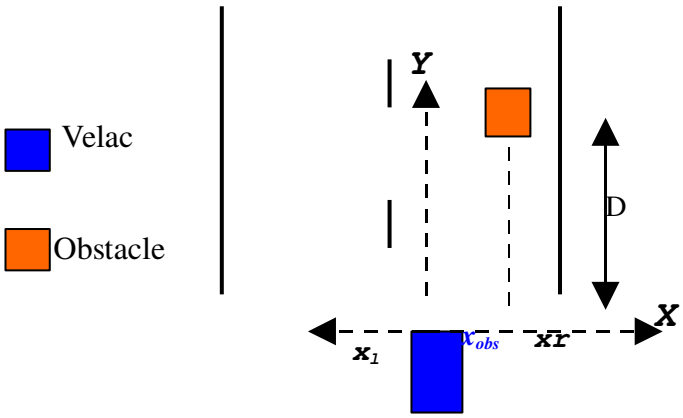


Figure 47 - Positionnement des obstacles sur la route.

Dans le projet ARCOS, la fonction « gérer les interdistances » a pour objectif d'appliquer la réglementation concernant le temps minimum de suivi (2s). On calcule donc la vitesse relative *Vr* et on en déduit le temps à collision *ttc*.

$Vr = sqrt(Vrx^2 + Vry^2)$

$ttc = D/Vr$

Le conducteur sera averti de la non conformité de sa conduite si :

$Vr < 0$

et

$|ttc| < 2\ sec$

4.4 Suivi des obstacles

4.4.1 Algorithme de vision pour la phase de suivi

Le télémètre laser fournit des données (état des obstacles) toutes les 400 ms. Cette période est trop longue pour la fonction interdistance dans laquelle il est nécessaire de gérer des insertions et désinsertions de véhicule à vive allure. Afin de sur-échantillonner ces données, nous proposons un système de poursuite par vision. Il permet de mettre à jour l'état des véhicules toutes les 40 ms.

Pour suivre les obstacles on utilise un algorithme de segmentation des images. Cet algorithme est basé sur l'observation selon laquelle une voiture vue de derrière présente des segments horizontaux en grand nombre. Le modèle est donc un rectangle englobant de tels segments. L'algorithme consiste en les étapes suivantes:

1. Obtention de l'image.
2. Création d'une fenêtre d'intérêt autour de l'obstacle.
3. Segmentation d'image dans cette fenêtre.
4. Mesure des valeurs extrémales des segments.
5. Construction des rectangles autour des segments.
6. Comparaison des rectangles avec des modèles prédéterminés. Recherche du rectangle dont les dimensions correspondent au mieux au véhicule suivi.
7. Calcul du centre de ce rectangle (centre de l'obstacle).

Ce processus est réalisé dans des fenêtres d'intérêts ce qui réduit les difficultés et permet à un tel algorithme, relativement succinct, d'être opérant (Figure 48). Les ombres portées peuvent cependant constituer une difficulté.

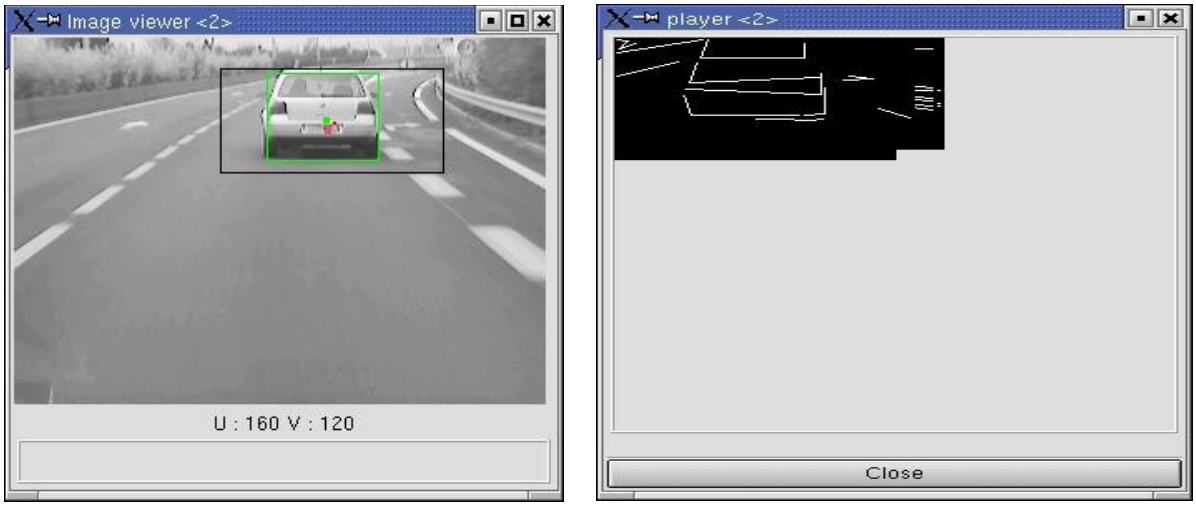


Figure 48 - Détection par segmentation. La figure (a) montre la fenêtre d'intérêt (noir) centrée sur le centre de l'obstacle (point-vert) et le rectangle autour de l'obstacle détecté (rectangle-vert).Figure (b) montre les différents rectangles trouvés par la segmentation.

En effet, l'algorithme de segmentation considère classiquement l'ombre comme une partie de la voiture. Ainsi il fusionne les segments des ombres avec ceux de la voiture et donne un rectangle englobant plus grand que la taille normale d'obstacle. Ceci peut conduire à des résultats erronés. Dans l'algorithme de segmentation nous trouvons le plus grand rectangle possible en utilisant les valeurs des segments horizontaux (u_{init}, u_{fin}) et verticaux (v_{init}, v_{fin}) pour trouver u_{min}, v_{min}, u_{max} et v_{max}.

Donc le rectangle est donné par :

Largeur = $u_{max} - u_{min}$; où u_{min}=min[$u1_{init}$,$u2_{init}$,$u3_{init}$.....un_{init}] et
u_{max}=max [$u1_{fin}$,$u2_{fin}$,$u3_{fin}$.....un_{fin}]

Longueur = $v_{max}-v_{min}$; où v_{min}=min[$v1_{init}$,$v2_{init}$,$v3_{init}$.....vn_{init}] et
v_{max}=max [$v1_{fin}$,$v2_{fin}$,$v3_{fin}$.....vn_{fin}]

Quand il y a une ombre, les segments horizontaux de l'ombre sont ajoutés et donc nous trouvons un rectangle plus grand (la largeur est beaucoup plus grande). Pour s'affranchir de cette difficulté, l'alternative consiste à corriger ce rectangle en le recréant autour du barycentre.

La largeur de ce nouveau rectangle est deux fois le distance minimum entre le barycentre et le coté (voir Figure 49 et Figure 50).

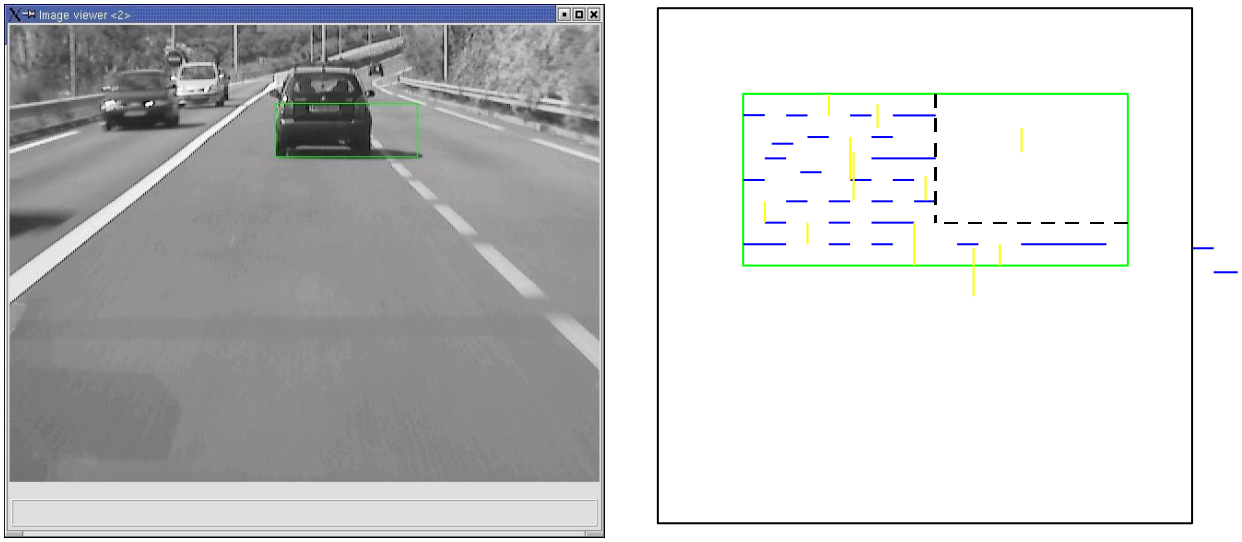


Figure 49 - Effet de l'ombre dans la segmentation. Les segments horizontaux de l'ombre sont insérés dans le rectangle englobant.

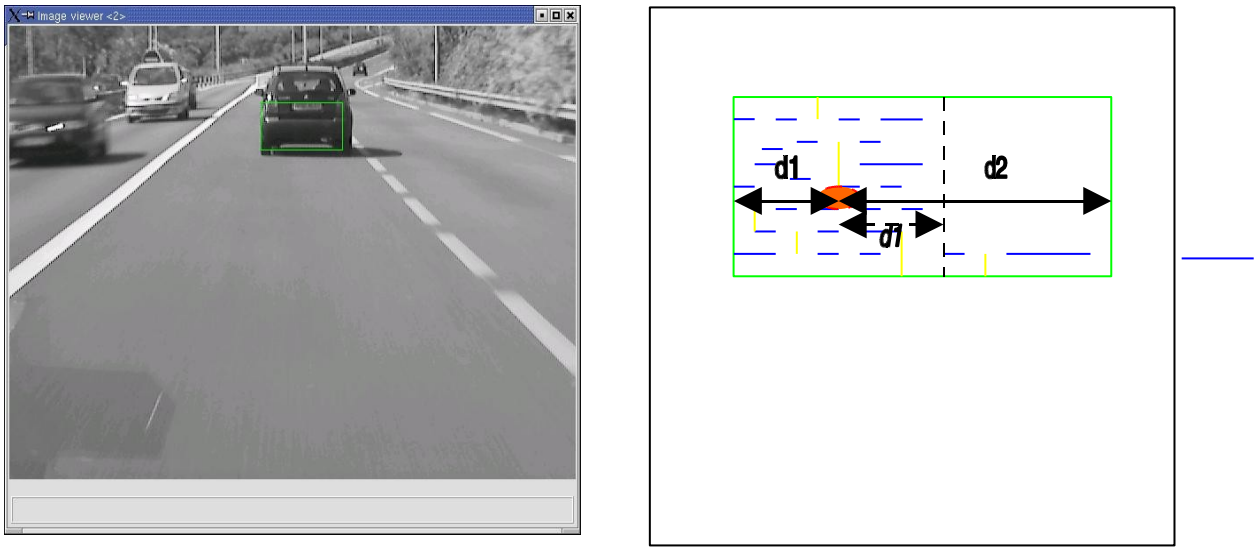


Figure 50 - Correction du rectangle englobant avec le barycentre. Figure (b) montre la correction avec le barycentre (point-rouge).

4.4.2 Calcul de l'état de l'obstacle

Le calcul de l'état de l'obstacle fait classiquement l'objet d'un filtrage. Cette estimation doit être maintenue pendant les phases de suivi par vision. Nous avons donc utilisé un modèle à vitesse constante dans lequel l'accélération est identifié au bruit. Les mesures intègrent la profondeur et la position de l'obstacle dans l'image.

Vecteur d'état : $X = [\ x\ \ x'\ y\ \ y'\ z\ \ z'\ ]^t$
Vecteur de mesure : $Z = [\ u\ v\ y\ ]^t$

L'estimation de Kalman est donnée par :

$$X_{1/1} = X_{0/0} + K.[\ \text{Innovation}\ ]$$

où l' innovation est donnée par innovation = $[\ Z - h(X_{1/0})\ ]$ et le gain $K = P_{1/0}\ .\ [\ H\ .\ P_{1/0}\ .\ H^t + R\ ]^{-1}$
$P_{1/0}$ est la matrice de covariance de la prédiction et H le jacobien de la matrice de mesure h(X).

La covariance estimée est donnée par :

$P_{1/1} = [\ I - K\ H].\ P_{1/0}$ où I est la matrice identité

Comme la profondeur n'est pas mesurée par les données télémétriques lors de la phase de suivi, c'est la mesure de la largeur du véhicule dans l'image qui permet de remonter à cette information. Dans ce cas on utilise donc le rectangle englobant les segments dans l'image. En effet, quand les données télémétriques sont disponibles, on connaît la taille de l'obstacle. On est donc en mesure de déduire ensuite sa distance à partir de sa taille dans l'image.

Entre chaque acquisition télémétrique (toutes les 400ms), on suppose que la largeur de l'obstacle est constante (objet non déformable). Pendant cette période, on utilise la taille de boite que l'on a trouvée avec la segmentation à chaque image (40 ms) et la largeur connue de l'obstacle pour faire le calcul de la distance de l'obstacle y.

- Données télémétriques : (calcul largeur d'obstacle avec les données télémètre et données image)
  - largeur_obstacle(3d)
  - distance_obstacle(3d)

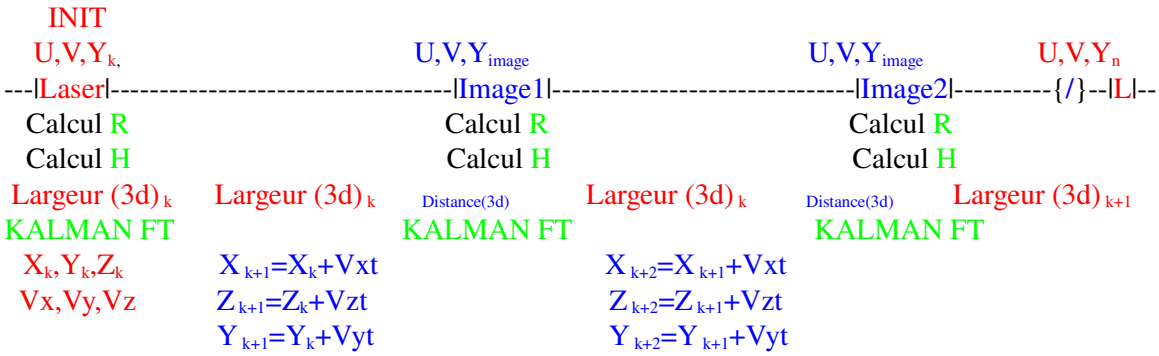
- Données image : calcul de la distance de l'obstacle(3d) avec largeur d'obstacle (supposée constante) et données image)
  - distance_obstacle(3d)= f (largeur_obstacle(3d), taille_boîte(2d))
  - distance_obstacle(3d)= f(largeur_obstacle(3d) , taille_boîte(2d))

  - distance_obstacle(3d)= f(largeur_obstacle(3d) , taile_boîte(2d))

- Données télémétriques: (calcul nouveaux largeur d'obstacle.....)
  - largeur_obstacle(3d)
  - distance_obstacle(3d)

Où la fonction f dépend de la projection des points 3d dans l'image 2d.

Algorithme



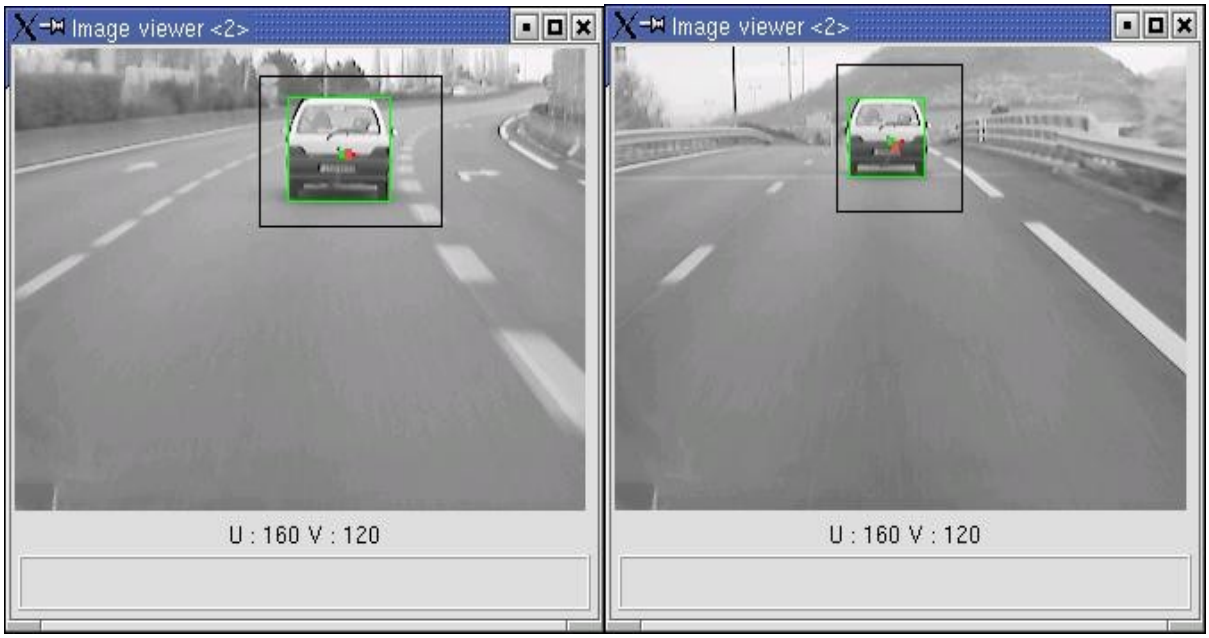


Figure 51 - Mesure de la distance en utilisant le taille des boites (rectangle englobant).Figure (a) et (b) montre deux instances où le même obstacle est détecté à différentes distances donc la taille du rectangle englobant est différente aussi. L'obstacle dans l'image (b) est plus loin que dans image (a).

4.4.2.1 Résultats

La Figure 52 montre que la distance et la vitesse de l'obstacle dans le repère Velac (x, y, z ,vx, vy, vz) sont très bien estimées (le bruit bien filtré). La Figure 53 montre que le résultat converge très rapidement (<10 itérations) avec une erreur entre 0,0008 et 0,4 m pour les distances (x,y,z) et entre 0 et 0,3 m/s pour les vitesses (vx, vy, vz). On ne trouve pas une grande différence entre la valeur estimée

et la vraie valeur de Y. Ces données ont été obtenues pendant les tests de suivi d'un véhicule lors des essais d'implémentation des fonctions ARCOS.

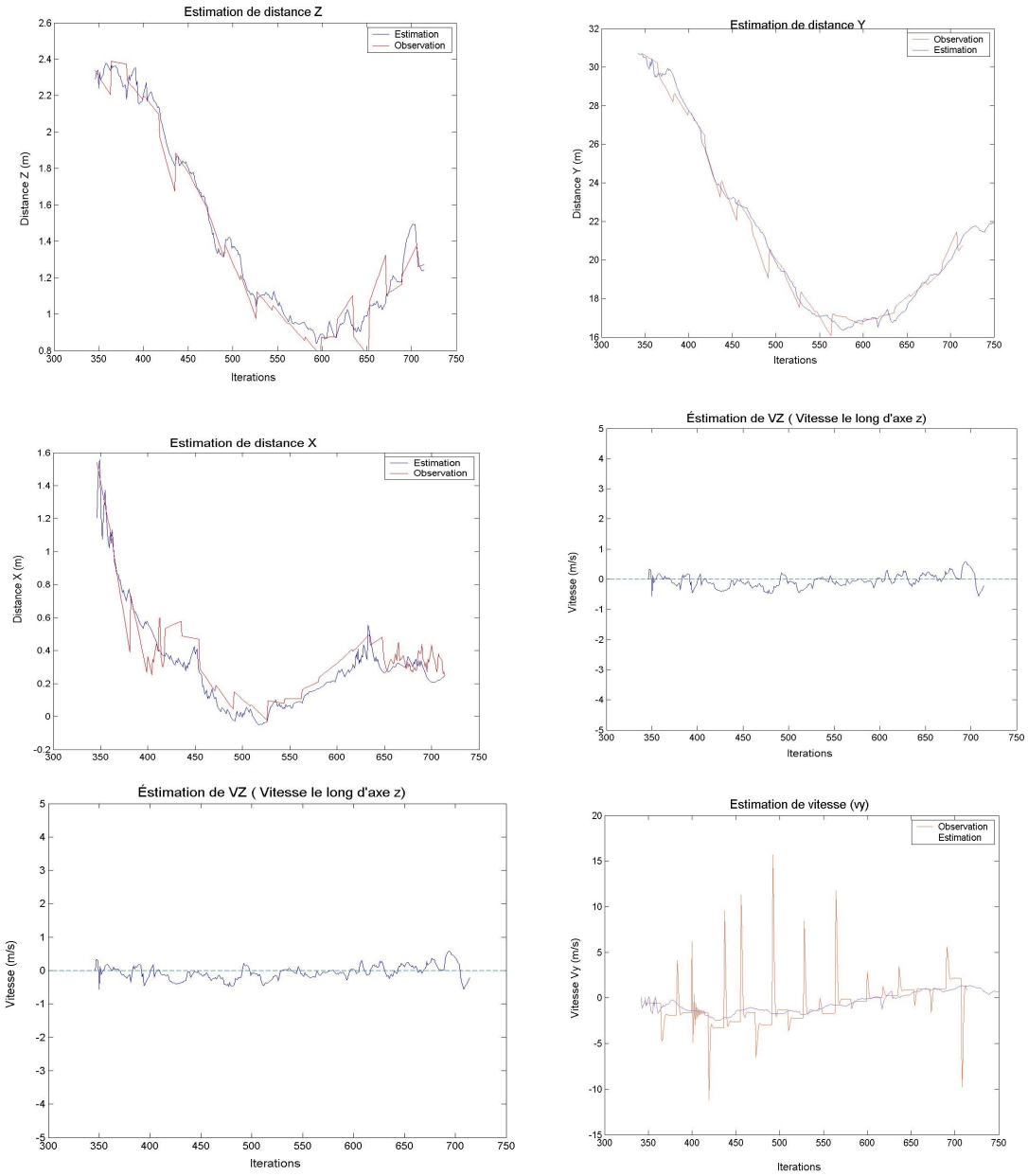


Figure 52 - Résultats du filtre de Kalman (Estimations sur distance et vitesse).

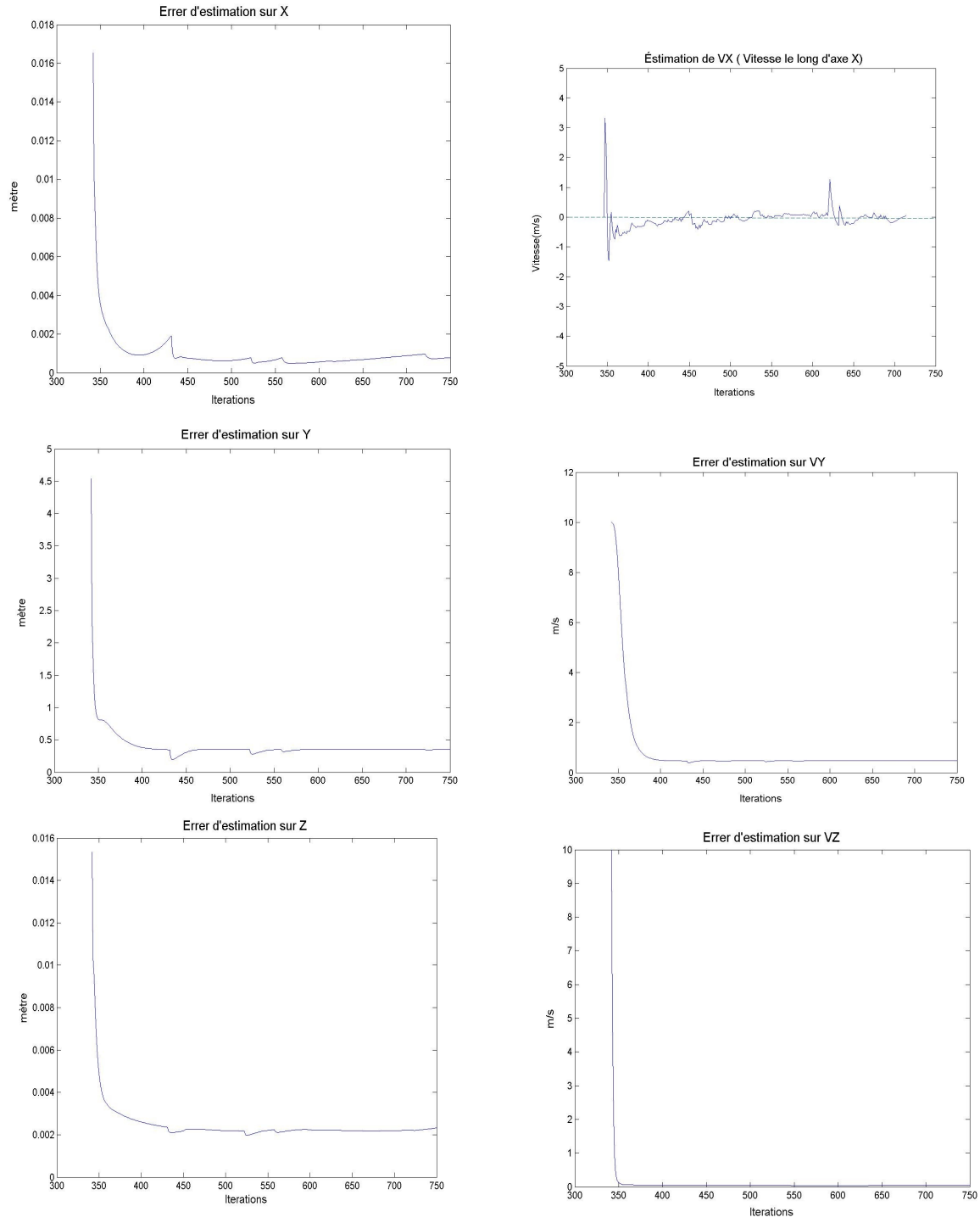


Figure 53 - Résultats du filtre de Kalman (Erreur d'estimations sur distance et vitesse).

4.4.2.2 Suivi de l'état de plusieurs obstacles

Le cahier des charges de la fonction « gérer les interdistances » permettrait de ne suivre que le véhicule immédiatement devant le véhicule expérimental et se trouvant dans la même voie de circulation. Cependant, afin d'anticiper, dans les meilleurs délais, les insertions d'autres véhicules dans la voie de circulation ou un éventuel dépassement du véhicule suivi, il est nécessaire de suivre l'ensemble des véhicules présents à l'avant du véhicule expérimental. L'ensemble des véhicules détectés sur les voies adjacentes et sur la voie utilisée fait donc l'objet d'un traitement analogue à celui décrit précédemment.

Figure 54 - Suivi d'obstacles multiples. Figure (a) montre les centres des obstacles suivis et (b) montre le rectangle englobant chacun des obstacles.

4.5 Conclusion

L'algorithme décrit dans ce chapitre a été spécifiquement mis au point et implémenté pour satisfaire les exigences de la fonction « gérer les interdistances ». Son architecture générale permet l'insertion d'un processus de vision quelconque capable de suivre un véhicule à partir d'une position initiale délivrée par le processus de reconnaissance 3D élaboré dans l'image télémétrique. Le processus peut ainsi exploiter la connaissance des dimensions du véhicule suivi. En retour, il permet un sur-échantillonnage de la cinématique fournie par les seuls traitements des images de profondeur qui n'opèrent pas à une cadence suffisante pour cette application. Le processus de vision mis en œuvre va au-delà de la simple validation du schéma global puisqu'il permet d'obtenir des résultats fiables qui ont été validés dans le cadre du projet ARCOS.

5 Modifications spécifiques du logiciel DBITE

Le logiciel DBITE (Driver Behaviour Interface Test Equipement) permet l'acquisition, le traitement temps réel ou le rejeu de données datées issues de divers capteurs.
Le projet ARCOS a vu la mise en place d'une nouvelle version du logiciel DBITE, que nous prenons l'habitude d'appeler DBITE_graph en référence à la structure arborescente sur laquelle repose le principe de cette nouvelle version, et sur laquelle nous reviendrons. Avant cela, insistons sur les apports de cette nouvelle version.

5.1 Présentation du DBITE

5.1.1 Installation du DBITE

Les implémentations de fonctions Arcos ont nécessité de nouvelles procédures d'installation qui ont essentiellement trait aux nouveaux matériels devant être utilisés.

5.1.1.1 Problèmes liés à l'IEEE1394

Des problèmes peuvent se produire avec l'IEEE1394. Plusieurs solutions pour les résoudre sont proposées ci-dessous :

- prendre la libraw 10.0 (ou postérieure)
- dans le fichier config_official, vérifier que le chemin de la librairie libraw1394 est correct.

Exemple :
DEFINES {
__CAMERA1394_ = /usr/local/lib/libraw1394.so

- charger le module raw1394 : modprobe raw1394

Pour éviter d'effectuer cette opération à chaque redémarrage, faire en sorte que cette librairie soit insérée automatiquement :
sous la Suse 9.1, ajouter dans le fichier /etc/modprobe.conf
"alias char-major-171 raw1394"
sous la Mandrake 9.2, cela s'est fait automatiquement.

- ajouter "/usr/local/lib" (ou autre selon le chemin de la libraw) dans le fichier /etc/ld.so.conf, puis taper ldconfig

- vérifier les droits d'accès sur le fichier /dev/raw1394

5.1.1.2 Module du CAN

Pour le module du CAN, nous l'avons modifié lors du passage de la Mandrake 9.2 vers la Suse 9.1 (noyau 2.4 -> 2.6). Il s'agit donc d'utiliser le bon selon le noyau utilisé. Pour installer ce module, voici comment procéder :

Sous la Mandrake 9.2 :

```
make ano
make install_ano
./install_can_ano.sh
```

Puis pour inclure ce module au démarrage, c'est ce même script qui doit être appelé : ajouter dans le fichier /etc/rc.local : “/usr/local/bin/install_can_ano.sh”

Sous la Suse 9.1 :

```
make
make install
./install_can_ano.sh
```

Puis pour inclure ce module au démarrage, ce n'est plus le script qui doit être appelé, on se contente d'insérer le module. Donc ajouter dans le fichier
/etc/inittab : “can:2345:wait:/sbin/insmod /usr/local/modules/can_ano.ko”

Si le module ne fonctionne pas, vérifier les points suivants :
- le module a bien été inséré : /dev/canbus et /dev/can1 existent

Trois exécutables insérés dans le répertoire /bin peuvent également être utiles :
–canread : pour vérifier les différentes trames passant sur le CAN
–canread_gyro : la même chose mais avec en plus l'affichage à intervalles réguliers de la valeur moyenne du zéro du gyromètre.
–canwrite : pour envoyer une trame sur le CAN

Si l'on souhaite recompiler ces exécutables, voici comment procéder :

```
gcc read_2can.c -o canread -lpthread
gcc gyro.c -o canread_gyro -lpthread
gcc write_can.c -o canwrite -lpthread
```

Si on souhaite installer le DBITE pour l’utiliser uniquement en Player, il n’est bien sûr pas nécessaire d’installer le module du CAN. Cependant, il faudra créer un répertoire /usr/local/include/can et y copier le fichier can.h qui se trouve dans le DBITE sous Modules/Can_pci.

5.1.2 Librairies nécessaires

Une interface OpenGL a été inclue dans le DBITE pour l’interface ARCOS. Plusieurs librairies spécifiques doivent donc êtres installées (ces librairies sont normalement disponibles sur les CD de la distribution Linux utilisée) :
. gtkglarea : permet l'utilisation de fonctions OpenGL dans une fenêtre GTK.
. GLU : une couche OpenGL
. GLUT : une autre couche OpenGL

Deux interfaces permettant un rendu assez complet des signaux transitant sur le CAN (angle au volant, vitesse, clignotants, ...) ont également été inclues. Ces interfaces utilisent la librairie . libgnome
Des difficultés peuvent être rencontrées lors de la compilation du DBITE, concernant cette librairie. Plusieurs solutions ont été apportées selon la distribution Linux utilisée. On liste les problèmes pouvant être rencontrés :

- une erreur sur le fichier gnomesupport.h : dans /usr/include/gnome-1.0, faire un lien vers /usr/lib/gnome-libs/include/gnomesupport.h
- une erreur de compilation dans le fichier libintl.h : dans /usr/include/gnome-1.0/libgnome/gnome-i18n.h, commenter toute la partie #ifdef ENABLE_NLS.

Dans le cadre d'ARCOS, une interface QT était nécessaire (répertoire IHM_ARCOSL2). L'intégration de cette interface dans le DBITE se révélant problématique (cohabitation QT et GTK difficile), un système de mémoire partagée a été mis en place. On consultera le chapitre concernant cette interface pour avoir davantage de précisions.

Le son a également été ajouté dans le DBITE. Pour l'exploiter, il faut installer la librairie fmod, que l'on pourra trouver dans le répertoire /lib du DBITE. Son installation se résume à :

- copier le fichier fmod.h dans le répertoire /usr/include
- copier le fichier libfmod.so dans /usr/lib

Sur certaines versions du DBITE, on a également ajouté un algorithme développé par Nexyad. Pour

une bonne compilation de cet algorithme :

- installer wx_gtk (normalement disponible sur le CD de la distribution, sinon, le télécharger (ou le récupérer dans le répertoire de suvagardes d'ARCOS), et recompiler :

```
/configure –with-gtk
make
su <mot de passe root>
make install
ldconfig
```

- copier le fichier libdetection_nexyad.so.0.0 dans /usr/local/lib
- faire le lien libdetection_nexyad.so.0 vers libdetection_nexyad.so.0.0
- faire le lien libdetection_nexyad.so vers libdetection_nexyad.so.0

Si malgré cela la compilation ne s'effectue toujours pas, et si cet algorithme n'est pas souhaité, voici comment procéder pour le supprimer du DBITE (et inversement si on souhaite l'ajouter):

- supprimer les fichiers algo_detection_nexyad.cc, algo_detection_nexyad.h et detection_nexyad.h qui se trouvent dans le répertoire Algorithms/DetectionVision/
- supprimer les lignes dans generic_algo.cc se rapportant à cet algorithme :

```
#include "DetectionVision/algo_detection_nexyad.h"
if (!strcmp(label, "detection_nexyad"))     { return true; }
if (!strcmp(label, "detection_nexyad"))     { return new
detection_nexyad_t(); }
```

- supprimer -ldetection_nexyad -lwx_gtk2-2.4 dans le config_official.txt

5.1.2.1 Installation du DBITE

Revenons maintenant à l'installation du DBITE :

- commencer par créer le Makefile :
  ./autom.pl config_offical.txt

- puis compiler : make

Normalement, le DBITE est prêt à l'emploi.

Une remarque également concernant la compilation de tous les programmes (répertoire Pgms) utilisés pour une exécution en live (Launcher) du DBITE. Auparavant, tous ces programmes étaient recompilés à chaque nouvelle compilation du DBITE. Ainsi, une modification même mineure sur le code entraînait des temps de compilation interminables. Ceci a été changé. Ces programmes sont maintenant recompilés uniquement si eux-mêmes sont modifiés (on peut par exemple faire un « touch » sur lefichier source du programme à recompiler). Autrement dit, lors d'une utilisation en live du DBITE, bien garder à l'esprit qu'une modification d'un algorithme (par exemple) n'a pas de répercussions sur l'exécutable. Il faut faire en sorte que cet exécutable soit recompilé pour que la modification prenne effet.

Voyons maintenant les modifications apportées au DBITE.

5.1.3 Modification apportée au DBITE

5.1.3.1 Arrêt du DBITE

Lors d'une utilisation en live du DBITE, l'arrêt normal consiste à appuyer sur le bouton STOP de l'interface graphique. Un problème récurrent était alors un phénomène de “freeze” du pc. Pour le résoudre, on a apporté la modification suivante au DBITE :
dans le RoadIO de la Caméra, fichier io_cam1394.cc, dans la fonction unset_iso_handler, on a commenté la ligne raw1394_stop_iso_rcv(handle, channel). Ceci a permit de résoudre le problème.

On s'est également aperçu qu'un arrêt du DBITE par la pression des touches CTRL-C pose certains problèmes : dans le cas où l'on procédait à l'enregistrement d'acquisitions, un tel arrêt avait pour conséquence une mauvaise fermeture des fichiers d'acquisitions, ce qui se traduisait par des Warnings affichés sur la console peu agréables lors de leur relecture. La solution apportée a consisté à modifier la gestion des signaux dans le répertoire Launcher pour qu'un arrêt du DBITE par CTRL-C ferme correctement les fichiers d'acquisitions. Les modifications apportées l'ont été dans :
Launcher/main.cc, fonction main()
Libraries/Tools/mysignal.cc, fonctions myhandler() et setEndFunction()

5.1.3.2 Ajout d'une méthode closeViewer()

Nous avons ajouté une méthode closeViewer() à la classe generic_viewer_t. Cette méthode, par défaut, ne fait rien, mais peut-être redéfinie si besoin est.

Dans le cas d'une utilisation du DBITE par Player, l'utilisation de cette méthode est transparente pour l'utilisateur, car tout a été mis en place pour qu'elle soit appelée automatiquement à la fin de l'utilisation du DBITE (changements dans le répertoire Player_tree).

En revanche, dans le cas d'une utilisation avec le Launcher, l'appel à cette méthode doit être explicitement incorporé au programme (répertoire Pgms) si l'appel à cette fonction est effectivement requis pour le viewer en question. Cet appel se fait dans la fonction :

```
void end(void *p)
{
       r_input->closeIO();
       r_output->closeIO();
       my_viewer->closeViewer();
       end_flag_video = true;
}
```

5.1.3.3 Ajout d'un fichier contenant le chemin du DBITE

Pour des raisons de confort d'utilisation, il était important de ne pas restreindre le répertoire contenant le code du DBITE à s'appeler obligatoirement DBITE ou DBITE_graph. Cependant, il arrive que dans le code, on soit obligé de faire référence à un chemin absolu pour un fichier (par exemple une image), et donc de connaître le nom que l'on a donné au répertoire du DBITE. C'est pour cette raison que l'on a modifié le script perl qui est executé lors de l'installation du DBITE :
./autom.pl config_official.txt
Désormais, à l'intérieur de ce script, on crée, à la racine du compte utilisateur, un fichier appelé Chemin_DBITE.txt, contenant le nom du répertoire du DBITE. Par exemple : /home/olivier/DBITE_graph

Désormais, lorsque l'on veut ouvrir un fichier contenant ce chemin, on procède de la manière suivante (exemple repris du viewer OpenGL pour ouvrir le fichier de texture de la route):

```
// chemin du compte utilisateur
    char *home = getenv("HOME");

// chemin du fichier Chemin_DBITE.txt
    char chemindbite[100];
    char cheminfichier[100];
    sprintf(cheminfichier, "%s/%s", home, "Chemin_DBITE.txt");

// lecture du contenu du fichier Chemin_DBITE.txt (ie le chemin du
répertoire DBITE)
    FILE * fp = fopen(cheminfichier, "r");
    if (!fp) printf("fichier Chemin_DBITE.txt introuvable\n");
    fscanf(fp, "%s", &chemindbite);
    fclose(fp);

 // Chargement de la texture du bitume
    char texture1[100];
    sprintf(texture1,"%s/%s",chemindbite,
Libraries/Viewers/RawImages/bitume_128 .JPG");
```

5.1.3.4 Ajout d'un fichier de configuration du DBITE

Un autre fichier a été ajouté pour un fonctionnement simplifié du DBITE. Il s'agit d'un fichier se nommant DonneesDBITE.txt et contenant un certain nombre de données utilisées dans le DBITE.

Pour lire une donnée contenue dans ce fichier, on appelle la fonction
float lireDonneeFloat(char * donnee) ou

int lireDonneeInt(char * donnee) selon le type de la donnée à lire, où donnee = l'intitulé de la donnée à rechercher dans le fichier DonneesDBITE.txt .
Ces fonctions sont contenues dans le fichier Libraries/Tools/ouvreFichierDonnees.h

5.1.4 L'interface QT

Cette interface QT a été mise en place dans le cadre d'ARCOS pour fournir une IHM commune à chacun des véhicules de démonstration. Elle nous a été fournie par le LIVIC. Cependant, sa mise en place sous Linux a nécessité quelques retouches. Le problème rencontré au cours de ces retouches est le suivant :
Cette interface a été développée sous QT Designer. Mais les retouches que nous y avons apporté ont porté directement sur le code. Or lors de la recompilation, certains fichiers sont effacés puis réécrits entièrement. C'est le cas en particulier des fichiers IHM_ARCOS.h et IHM_ARCOS.cpp, fichiers qui ont fait l'objet des retouches mentionnées au-dessus. Si l'on souhaite recompiler entièrement cette interface QT, voici la marche à suivre :

–effectuer une première compilation qui va sortir des messages d'erreur
make

–afficher les fichiers cachés, et ouvrir le répertoire .ui/

–ouvrir les fichiers .ui/IHM_ARCOS.cpp et .ui/IHM_ARCOS.h, supprimer tout leur contenu

–remplacer ce contenu par le code contenu dans les fichiers SaveFic/IHM_ARCOS.cpp et SaveFic/IHM_ARCOS.h

–effectuer une nouvelle compilation
make

Remarque : il est absolument nécessaire, dans l'étape 4, de supprimer le code pour le remplacer par le nouveau. On pourrait penser qu'il revient au même d'écraser les anciens fichiers par les nouveaux, mais ce n'est pas le cas.

Cette interface QT lance le DBITE lorsque l'on clique sur le bouton Enregistrer , via des scripts, comme mentionné dans la notice d'utilisation de l'interface (IHM_ARCOS.doc). Reste alors à faire communiquer le DBITE avec cette interface. Tout se fait à travers une mémoire partagée, que l'on a choisi de déclarer dans un viewer, puisque l'interface QT est au final un viewer. Voyons ceci à travers les fichiers Viewers/Arcos/Anticollision_MP.cc (et .h) (même chose pour Interdistance_MP.cc) :

```
// variables (dans le .h)

// classes définies dans les fichiers IHM_MP.h et MP.h (fournis par
le LIVIC)

IHM_MP_Struct *MPIStruct, *MPIStruct2;

MP *IHM_Appli;


// Initialisations :


// la structure dans laquelle on remplit les nouvelles informations

MPIStruct=(IHM_MP_Struct *)malloc(sizeof(IHM_MP_Struct));
```

```
// la structure tranmise par la mémoire partagée
MPIStruct2=(IHM_MP_Struct *)malloc(sizeof(IHM_MP_Struct));


// la variable de mémoire partagée
IHM_Appli=new MP();


// initialisation de la structure de transfert
MPIStruct2=(IHM_MP_Struct *)IHM_Appli-
>InitMP(false,CLEF_IHM_MP,sizeof(IHM_MP_Struct));
// ATTENTION :  CLEF_IHM_MP est définie dans le fichier IHM_MP.h par
:
// #define CLEF_IHM_MP "/home/olivier/IHM_ARCOSL2/IHMmain.cpp"
// si ce chemin devait être changé, il serait donc nécessaire de
modifier cette variable


// Exemple de mise à jour des informations :


MPIStruct->Anticollision_S.Etat=F_A;
MPIStruct->Anticollision_S.PType=Aucun;
sprintf(MPIStruct->Anticollision_S.Texte,"");


// Echange des données


// Prise du sémaphore de protection de la mémoire
IHM_Appli->MP_P();


// Recopie des informations dans la structure de transfert
memcpy(MPIStruct2,MPIStruct,sizeof(IHM_MP_Struct));


// Relâchement du sémaphore de protection de la mémoire
IHM_Appli->MP_V();
```

5.1.5 La documentation du DBITE

En complément de la présente notice, nous avons incorporé au DBITE une documentation par pages html générées par Doxygen. Cette documentation concerne les nouveaux RoadIO, Algorithmes et Viewers.
Voici comment opérer si l'on souhaite faire de même pour d'autres fichiers. On pourra se référer au DBITE pour avoir un exemple.

Voyons tout d'abord comment se présentent les commentaires Doxygen :

```
/** Exemple de phrase d'introduction pour décrire une classe
*  La suite de la description
*  \image html nom_du_fichier_image       si l'on souhaite insérer
une image
*/


/** Une méthode de la classe
```

```
 *@description d'un argument de la méthode
 *@description d'un autre argument
 *@retour de la classe
 */

attribut      /*!< un attribut de la classe*/
```

Puis voyons comment générer la documentation html pour un fichier mon_fichier.h

–Modifier le Makefile :
. ajouter en tête DOC = doxygen
. DOC_SOURCES = mon_fichier (sans l'extension !)
. all : $(OBJECTS) doc
. doc : ... ce qu'il faut pour générer la doc. On se réfèrera à un exemple déjà créé pour copier/coller cette partie

–Créer, si ce n'est pas déjà fait, un répertoire Doc

–Créer éventuellement un répertoire Images si l'on souhaite insérer des images dans la documentation html

–Copier/coller un exemple de fichier Doxyfile_mon_fichier.dox

–Ouvrir ce fichier .dox avec Doxygen Wizard et modifier les champs qui ont besoin de l'être :

Project_name
Input (ATTENTION : ne mettre que des chemins relatifs : ./mon_fichier.h)
Supprimer éventuellement le IMAGE_PATH s'il n'y a pas d'images
html_output html_mon_fichier

–Dans le répertoire contenant le makefile, exécuter “make doc”, ou bien recompiler entièrement le DBITE puisque la documentation est regénérée à chaque nouvelle compilation.

Certaines conventions doivent être respectées pour un bon fonctionnement de la documentation. On insiste en particulier sur le fait que tous les chemins doivent être en relatif, et qu'il est nécessaire de reprendre le nom du fichier à documenter sans son extension dans les différents champs de Doxygen Wizard et dans le Makefile. On se référera aux nombreux exemples déjà documentés pour apprécier ces conventions.

5.2 Utilisation du DBITE en Player

5.2.1 Avant de commencer

Il est important de noter une modification majeure dans l'utilisation du DBITE. Elle concerne les algorithmes et les viewers.

On rappelle que le prototype des fonctions appelées lors du déclenchement d'un algorithme ou d'un viewer sont:
- algorithmes : algo(unsigned char *buffer, int size, road_time_t t, void **param)
- viewers : display(unsigned char *buffer, int size, road_time_t t,void **param)

Les entrées de ces méthodes sont
- buffer : l'entrée principale de l'algorithme
- param : une liste d'entrées secondaires de l'algorithme.

La modification majeure porte sur la fonction check_change qui se trouve dans le fichier Player_tree/execution_graph. Auparavant, cette fonction était faite de telle manière qu'un algorithme ou un viewer s'active chaque fois qu'une de ses entrées (principale ou secondaire) changeait. On l'a modifiée de telle manière que cette activation ne se produise que lorsque l'entrée principale change. Les entrées secondaires ont alors pour valeur leur dernière valeur connue.

Expliquons les raisons de ce choix :
- la souplesse d'utilisation est beaucoup plus intéressante, puisque l'on n'est pas contraint à un fonctionnement de l'algorithme ou du viewer. A nous de choisir l'entrée principale qui va déclencher.

- cette utilisation est plus en rapport avec le fonctionnement du DBITE en live (Launcher). En effet, si l'on observe les programmes (répertoire Pgms), on se rend compte, par exemple, que le fonctionnement est basé sur une lecture d'une donnée capteur qui déclenche un algorithme et que les entrées secondaires sont récupérées.

- la charge pourrait être trop importante, par exemple dans le cas d'un algorithme utilisant la vidéo en entrée principale et l'odomètre (CAN) en entrée secondaire, puisque un déclenchement par l'odomètre conduirait à des appels incessants de l'algorithme.

5.2.2 Exemple possible pour ARCOS

5.2.2.1 Le fichier de configuration

Le fichier de configuration suivant peut être utilisé dans le cadre de la fonction interdistances. Chacune de ses composantes va faire l'objet d'une explication ci-dessous.

```
file : /home/olivier/Acquisitions/video.dbt
       buffer_out : video


file : /home/olivier/Acquisitions/telemeter.dbt
       buffer_out : telemeter


file : /home/olivier/Acquisitions/can.dbt
       buffer_out : can


file : /home/olivier/Acquisitions/modemradio.dbt
       buffer_out : modemradio


algo : dernieres_valeurs
       buffer_in : can
       buffer_out : can
       param_out : lastvalues
```

```
algo : dist_vit_acc
       buffer_in : can
       buffer_out : can
       param_out : donneescan

algo : can_transform_algo
       buffer_in : can
       buffer_out : can
       param_out : paramcan

algo : road_loc
       buffer_in : video
       buffer_out : video
       param_in : paramcan
       param_out : paramvideo

algo : detection_tracking_vision_gyro
       buffer_in : telemeter
       buffer_out : telemeter
       param_in : donneescan lastvalues paramvideo
       param_out : paramdetect

algo : detection_tracking_vision
       buffer_in : video
       buffer_out : bufseg
       param_in : paramdetect paramvideo donneescan lastvalues
       param_out : seg

algo : sauver_obstacles
       buffer_in : seg
       param_in : donneescan lastvalues modemradio

viewer : image_road_obstacles_viewer
       buffer_in : video
       param_in : paramvideo seg

viewer : image_obstacle_segmentation_viewer
       buffer_in : video
       param_in : bufseg

viewer : telemeter_arcos_viewer
       buffer_in : telemeter
       param_in : paramdetect

viewer : telemeter_viewer
       buffer_in : telemeter
       param_in : paramdetect
```

```
viewer : road_obstacles_viewer_gl
      buffer_in : video
      param_in : paramdetect paramvideo
```

5.2.2.2 Les entrées

Voici la syntaxe pour déclarer les entrées de l'algorithmes, c'est à dire les fichiers d'acquisitions.

```
file : /home/olivier/Acquisitions/video.dbt
      buffer_out : video

file : /home/olivier/Acquisitions/telemeter.dbt
      buffer_out : telemeter

file : /home/olivier/Acquisitions/can.dbt
      buffer_out : can

file : /home/olivier/Acquisitions/modemradio.dbt
      buffer_out : modemradio
```

5.2.2.3 Les algorithmes du CAN

Trois algorithmes sont appelés sur l'entrée CAN.

algo : dernieres_valeurs
Cet algorithme a pour objet de retenir pour chaque type de trame la dernière valeur reçue.

algo : dist_vit_acc
Celui-ci calcule la distance parcourue, la vitesse et l'accélération instantanée du véhicule.

algo : can_transform_algo
Cet algorithme a été réalisé spécialement pour l'algorithme de détection de lignes blanches. Son but est de convertir les trames en une structure de données Observation exploitable par l'algorithme de détection de lignes blanches. Ces observations sont ensuite insérées dans une liste, représentée sous le nom de paramcan dans le fichier de configuration.

5.2.3 Algorithme de détection de lignes blanches

Cette liste est exploitée par l'algorithme de détection de lignes blanches, qui lui-même insère de nouvelles observations dans cette liste. Ceci est possible, car le passage des paramètres se fait par adresse. Deux algorithmes peuvent donc “communiquer” par ce biais.

```
algo : road_loc                    // nom de l'algorithme
      buffer_in : video               // entrée principale de
l'algorithme
      buffer_out : video              // sortie principale de
l'algorithme
      param_in : paramcan             // entrée secondaire de
l'algorithme
```

```
      param_out : paramvideo           // sortie secondaire de
l'algorithme
```

On peut faire quelques remarques supplémentaires à ce niveau :

- le nom de l'algorithme (ou du viewer) est donné par ce que l'on a définit dans le fichier generic_algo.cc (ou generic_viewer.cc)

- les sorties de l'algorithme sont de deux au maximum. On leur a donné ici le nom de sortie principale et sortie secondaire, mais dans la pratique, leur rôle est le même (contrairement aux entrées pour lesquelles il faut bien distinguer entrée principale et entrées secondaires). A titre personnel, j'ai pris l'habitude, dans le cas où l'algorithme ne change pas l'entrée principale, de nommer de la même manière la sortie principale, et de considérer que la vraie sortie de l'algorithme est la sortie secondaire. Une exception toutefois, lorsque j'ai besoin de deux sorties, par exemple avec une variante de l'algorithme de détection de lignes blanches :

```
algo : road_loc_carte
       buffer_in : video
       buffer_out : carte
       param_in : paramcan
       param_out : paramvideo
```

Cet algorithme a en plus une sortie relative aux données gps (carte).

5.2.4 Algorithmes de détection d'obstacles

Deux algorithmes remplissent ce rôle, l'un venant à la suite de l'autre :

```
algo : detection_tracking_vision_gyro
       buffer_in : telemeter
       buffer_out : telemeter
       param_in : donneescan lastvalues paramvideo
       param_out : paramdetect


algo : detection_tracking_vision
       buffer_in : video
       buffer_out : bufseg
       param_in : paramdetect paramvideo donneescan lastvalues
       param_out : seg
```

Le premier est un algorithme de détection et de tracking d'obstacles travaillant principalement sur les données télémétriques (entrée principale : telemeter). Néanmoins, il a besoin pour fonctionner de données
provenant de l'algorithme de détection de lignes blanches
provenant du can :
- la sortie de l'algorithme de calcul de distance, vitesse, accélération
- la sortie de l'algorithme retenant les dernières valeurs pour chaque type de trame.

Profitons de cet exemple pour revenir sur la manière dont se déclenche un algorithme (5.2.1 Avant de commencer). On voit bien sur cet exemple qu'il est préférable que l'algorithme de détection d'obstacle

se déclenche à chaque nouvelle donnée télémétrique, plutôt qu'à chaque nouvelle entrée (télémétrique, vidéo ou can). En effet, c'est bien le télémètre qui permet la détection des obstacles. Les autres données sont là pour fournir des valeurs (position des lignes blanches, vitesse de notre véhicule) qu'il suffit d'aller chercher.

Le second algorithme est un algorithme de détection d'obstacle par vision, d'où l'entrée principale vidéo. Cet algorithme a comme entrées secondaires :
- la sortie de l'algorithme de détection d'obstacles par télémètre.
- la sortie de l'algorithme de détection de lignes blanches
- la sortie de l'algorithme de calcul de distance, vitesse, accélération
- la sortie de l'algorithme retenant les dernières valeurs pour chaque type de trame.

On remarque qu'il possède deux sorties :
- bufseg qui est utilisé pour une visualisation des résultats de cet algorithme.
- seg : qui est du même format que paramdetect. Le but de ceci est très simple : puisque les deux algorithmes de détection d'obstacles ont le même format de sortie, un autre algorithme ou un viewer qui aurait besoin de valeurs de détection d'obstacle pourra au choix venir à la suite du premier ou du second algorithme.

5.2.5 Sauver les obstacles

Illustration avec l'algorithme de sauvegarde des obstacles, des données CAN et des données modemradio (données envoyées par un véhicule tiers pour établir une réalité de terrain dans le cadre d'ARCOS). Voici comment est déclaré cet algorithme :

```
algo : sauver_obstacles
       buffer_in : seg
       param_in : donneescan lastvalues modemradio
```

On aurait très bien pu imaginer

```
algo : sauver_obstacles
       buffer_in : paramdetect
       param_in : donneescan lastvalues modemradio
```

selon que l'on désire enregistrer la détection d'obstacle par télémètre ou par vision.

A noter également que l'entrée principale de cet algorithme n'est pas une donnée capteur, mais le résultat d'un autre algorithme.

5.2.6 Les viewers

Le principe de fonctionnement des viewers est le même que celui des algorithmes avec une entrée principale et une ou plusieurs entrées secondaires.

```
viewer : image_road_obstacles_viewer
       buffer_in : video
```

```
       param_in : paramvideo seg

viewer : image_obstacle_segmentation_viewer
       buffer_in : video
       param_in : bufseg

viewer : telemeter_arcos_viewer
       buffer_in : telemeter
       param_in : paramdetect

viewer : telemeter_viewer
       buffer_in : telemeter
       param_in : paramdetect

viewer : road_obstacles_viewer_gl
       buffer_in : video
       param_in : paramdetect paramvideo
```

Ce dernier viewer est une interface utilisant OpenGL dans une fenêtre GTK.

5.3 Exemple d'un algorithme de simulation de brouillard

5.3.1 Le fichier de configuration

Voici l'exemple d'un fichier de configuration pour la simulation du brouillard sur une image vidéo. Cet exemple est intéressant car on souhaite pouvoir utiliser l'image altérée comme une image normale, en particulier ici l'afficher dans un viewer dédié à la vidéo.

```
file : /home/olivier/Acquisitions/video_od3.dbt
       buffer_out : video

file : /home/olivier/Acquisitions/can_od3.dbt
       buffer_out : can

algo : dernieres_valeurs
       buffer_in : can
       buffer_out : can
       param_out : lastvalues

algo : brouillard
       buffer_in : video
       buffer_out : brouillard
       param_in : lastvalues

viewer : image_viewer
       buffer_in : brouillard
```

On pourra s'étonner que le CAN intervienne dans cet algorithme, mais la raison est simple : la simulation de brouillard prend effet lorsque le conducteur de la voiture appuie sur un bouton au volant. Le signal de ce bouton est enregistré par le CAN.

5.3.2 Code

Voici une partie de la déclaration de la classe brouillard :

```
class brouillard_t : public generic_algo_t
{
	public :

	int initAlgo(bool own_buffer, void *ptr);
	void algo(unsigned char *buffer, int size, road_time_t t, void
**param=NULL);
	char *getTitle() { return "brouillard"; }

	unsigned char *getBuffer()
	{
		return FogBuffer;
	}

	unsigned char *getParamInit()
	{
		return (unsigned char *)(&ih);
	}

	private :
	image_header_t ih;
	int dimx;
	int dimy;
	unsigned char * FogBuffer;
};
```

On retrouve bien sûr les traditionnelles fonctions initAlgo et algo. Il y a également la fonction getBuffer qui retourne la sortie de l'algorithme (ie l'image altérée). Mais le plus intéressant réside dans la fonction getParamInit dont le but est de faire que l'image altérée puisse être considérée comme n'importe quelle image provenant de la caméra, en récupérant l'image_header contenant des informations descriptives de l'image initiale. La variable ih est remplie dans la fonction initAlgo :

```
int brouillard_t::initAlgo(bool own_buffer, void *ptr)
{
	ih = *(image_header_t*)ptr;
	dimx = ih.image_width;
	dimy = ih.image_height;
	FogBuffer = (unsigned char *)malloc(dimx*dimy);

	return 0;
}
```

5.4 Utilisation du DBITE en Launcher

5.4.1 Le fichier de configuration

Contrairement à son utilisation en Player, le DBITE ne repose pas uniquement sur des fichiers de configuration pour une utilisation en live (Launcher). Ces fichiers existent bien, mais uniquement pour définir les entrées, les sorties (fichiers d'acquisition) et les programmes appelés. Exemple :

```
live

/*     Control camera */

exec Camera1394/viewer1394 straightaway
       input : camera1394 node 0

exec Camera1394/Camera_Control/camera_control straightaway

/*     Control telemeter     */

exec  Z210_telemeter/Control/tof   straightaway

/*    Acquisition    */

exec Arcos/pgm_interdistance_vision_out label Arcos
       input : telemeter
       input : camera1394 node 0
       input : can_pci /dev/canbus
       input : modemradio_livic /dev/ttyS1
       output : file localhost /mnt/rack_l/telemeter.dbt
       output : file localhost /mnt/rack_l/video.dbt
       output : file localhost /mnt/rack_l/can.dbt
       output : file localhost /mnt/rack_l/modemradio.dbt

exec Text/entry3
       output : file localhost /mnt/rack_l/text.dbt
```

On voit que ce fichier de configuration fait appel au programme Text/entry3 et au programme Arcos/pgm_interdistance_vision_out (répertoire Pgms). Ce programme prend en entrée le télémètre, la caméra, le can et le modemradio, et produit en sortie les fichiers d'acquisitions correspondant. C'est un tel exemple de programme que l'on se propose de décrire ici. Ce programme est l'équivalent en Launcher du fichier de configuration pour Arcos que l'on a vu en Player (section 2.2.1).

On déconseille cependant l'emploi d'un tel exemple car la charge Sauvegarde Des Données + Traitement Des Données est importante. On préfèrera utiliser uniquement un programme de sauvegarde (par exemple Arcos/acquisition_telemeter_video_can_modemradio) ou uniquement un programme de traitement en live.

On remarque également que ce fichier de configuration fait appel aux programmes de configuration de la caméra et du télémètre. Si pour ce dernier, cette étape est inutile, il n'en va pas de même pour la caméra pour laquelle on se doit de configurer ses paramètres (taille de l'image, zoom, ...) avant d'initialiser les algorithmes. Cependant, il n'est pas très agréable, dans l'utilisation du DBITE, de répéter ces opérations de paramétrage. C'est pourquoi, on peut ajouter les lignes suivantes au programme utilisé (répertoire Pgms), pour que ces opérations soient automatiques :

```
// gain automatique
dc1394_auto_on_off(((io_camera1394 *)r_input_video)->handle,
((io_camera1394 *)r_input_video)->node, FEATURE_GAIN, 1);


// shutter automatique
dc1394_auto_on_off(((io_camera1394 *)r_input_video)->handle,
((io_camera1394 *)r_input_video)->node, FEATURE_SHUTTER, 1);


// iris automatique
dc1394_auto_on_off(((io_camera1394 *)r_input_video)->handle,
((io_camera1394 *)r_input_video)->node, FEATURE_IRIS, 1);


// zoom de 700
dc1394_set_feature_value(((io_camera1394 *)r_input_video)->handle,
((io_camera1394 *)r_input_video)->node, FEATURE_ZOOM, 700);


// taille 640x480
dc1394_set_video_mode(((io_camera1394 *)r_input_video)->handle,
((io_camera1394 *)r_input_video)->node, MODE_640x480_YUV411);


// 30 images par seconde
dc1394_set_video_framerate(((io_camera1394 *)r_input_video)->handle,
((io_camera1394 *)r_input_video)->node, FRAMERATE_30);
```

Il faut ensuite déclarer et initialiser chacun des algorithmes et viewers utilisés. On se rapportera au code pour voir la manière de procéder. Nous allons nous intéresser ici au thread de la vidéo :

5.4.2 Les threads

```
void *acq_loop_video(void *ptr)
{
       int nb_read;
       void * tab1[2];
       void * tab2[4];
       void * tab3[2];


       while (!end_flag_video) {
              if ((nb_read=r_input_video->readIO(buffer_video, &tim))
< 0) {
                     road_perror("Video acquisition problem :");
                     break;
              }
```

```
                // algo road_loc
                tab1[0] = can_transform_algo->getParam();
                tab1[1] = NULL;
                road_loc->algo((unsigned char *)buffer_video, -1, tim,
tab1);


                // algo detection_tracking_vision
                tab2[0] = detection_tracking_vision_gyro->getParam();
                tab2[1] = road_loc->getParam();
                tab2[2] = dist_vit_acc->getParam();
                tab2[3] = dernieres_valeurs->getParam();
                detection_tracking_vision->algo((unsigned char
*)buffer_video, -1, tim, tab2);


                // algo sauver_obstacles
                tab3[0] =  dist_vit_acc->getParam();
                tab3[1] = dernieres_valeurs->getParam();
                sauver_obstacles->algo((unsigned char
*)(detection_tracking_vision->getParam()), -1, tim, tab3);


                if ((nb_read=r_output_video->writeIO(buffer_video,
tim)) < 0) {
                        road_perror("Video record problem :");
                        break;
                }


                buffer_video_ready=true;
        }
        return NULL;
}
```

Ce thread commence par la lecture d'une donnée vidéo :
nb_read=r_input_video->readIO(buffer_video, &tim))

Puis sont appelés tous les algorithmes dont l'entrée principale est la vidéo :

L'algorithme de détection de lignes blanches prenant en entrée principale la donnée vidéo (buffer_video) et en entrée secondaire la sortie de l'algorithme de transformation des données can, via le tableau tab1 (tab1[0] = can_transform_algo->getParam();)
Une remarque importante : on voit que l'on a tab1[1] = NULL;
La raison est que l'algorithme road_loc est susceptible de prendre une deuxième entrée secondaire, mais que dans la situation présente, cette deuxième entrée n'est pas utilisée. Autant, dans le cas du fichier de configuration du Player, cela ne posait pas de souçis, autant dans le cas d'une utilisation en Launcher, il faut absolument préciser que cette deuxième entrée n'est pas utilisée en forçant tab1[1] à la valeur NULL.

L'algorithme de détection d'obstacle par vision prenant en entrée principale la donnée vidéo (buffer_video) et en entrées secondaires via le tableau tab2 :
- la sortie de l'algorithme de détection d'obstacle par télémètre (tab2[0] = detection_tracking_vision_gyro->getParam();)
- la sortie de l'algorithme de détection de lignes blanches (tab2[1] = road_loc->getParam();)

- la sortie de l'algorithme de calcul de distance, vitesse, accélération (tab2[2] = dist_vit_acc->getParam();)
- la sortie de l'algorithme retenant les dernières trames CAN (tab2[3] = dernieres_valeurs->getParam();)

On place également dans ce thread l'algorithme sauver_obstacles qui a pour entrée principale la sortie de l'algorithme de détection d'obstacles par vision (dans le même thread, on vient de le voir), et pour entrées secondaires les sorties de deux algorithmes relatifs au CAN.

Enfin, ce thread se termine par la sauvegarde de la donnée vidéo dans le fichier d'acquisition : nb_read=r_output_video->writeIO(buffer_video, tim)).

Evidemment, de tels threads doivent également être mis en place pour le télémètre et pour le CAN.

5.4.3 Callbacks d'affichage

Un callback d'affichage regroupe tous les viewers utilisés pour chaque type d'entrée. Nous décrivons ici celui utilisé pour la vidéo.
Le principe est le même que pour les algorithmes, avec le passage des entrées secondaires via un tableau.

```
gint viewer_video_call(gpointer ptr)
{
       void * tab1[2];
       void * tab2[1];
       void * tab3[3];

       if (!buffer_video_ready) { return 1; }

       tab1[0] = road_loc->getParam();
       tab1[1] = detection_tracking_vision_gyro->getParam();
       my_viewer_video->display((unsigned char *)buffer_video,
buffer_video_size, tim, tab1);

       tab2[0] = detection_tracking_vision->getBuffer();
       my_viewer_video2->display((unsigned char *)buffer_video,
buffer_video_size, tim, tab2);

       tab3[0] = detection_tracking_vision_gyro->getParam();
       tab3[1] = road_loc->getParam();
       tab3[2] = NULL;
       my_viewer_gl->display((unsigned char *)buffer_video,
buffer_video_size, tim, tab3);

       buffer_video_ready=false;

       if (end_flag_video) { gtk_main_quit(); return 0; }
         return 1;     // si 0 => fin du timeout
}
```

Trois viewers sont appelés ici :

- my_viewer_video qui est le viewer pour la visualisation des lignes blanches et des obstacles sur la vidéo.

- my_viewer_video2 qui est le viewer pour la visualisation de la sortie de l'algorithme de détection des obstacles par vision.

- my_viewer_gl qui est le viewer OpenGL permettant la visualisation des lignes blanches et des obstacles, en 3D. On remarque ici l'affectation tab3[2] = NULL, pour la même raison que pour l'algorithme de détection des lignes blanches (voir plus haut), à savoir qu'il faut bien préciser que ce viewer est susceptible de recevoir une troisième entrée secondaire (en l'occurence la carte GPS) mais qu'on ne l'utilise pas ici.

Evidemment, de tels callbacks peuvent également être mis en place pour le télémètre et pour le CAN, si du moins on souhaite des viewers pour ces données.

5.5 Utilisation du DBITE lors des démonstrations du 28 octobre 2004

Intéressons nous maintenant à la manière dont nous avons utilisé le DBITE dans le cadre des démonstrations du 28 octobre 2004 pour ARCOS.

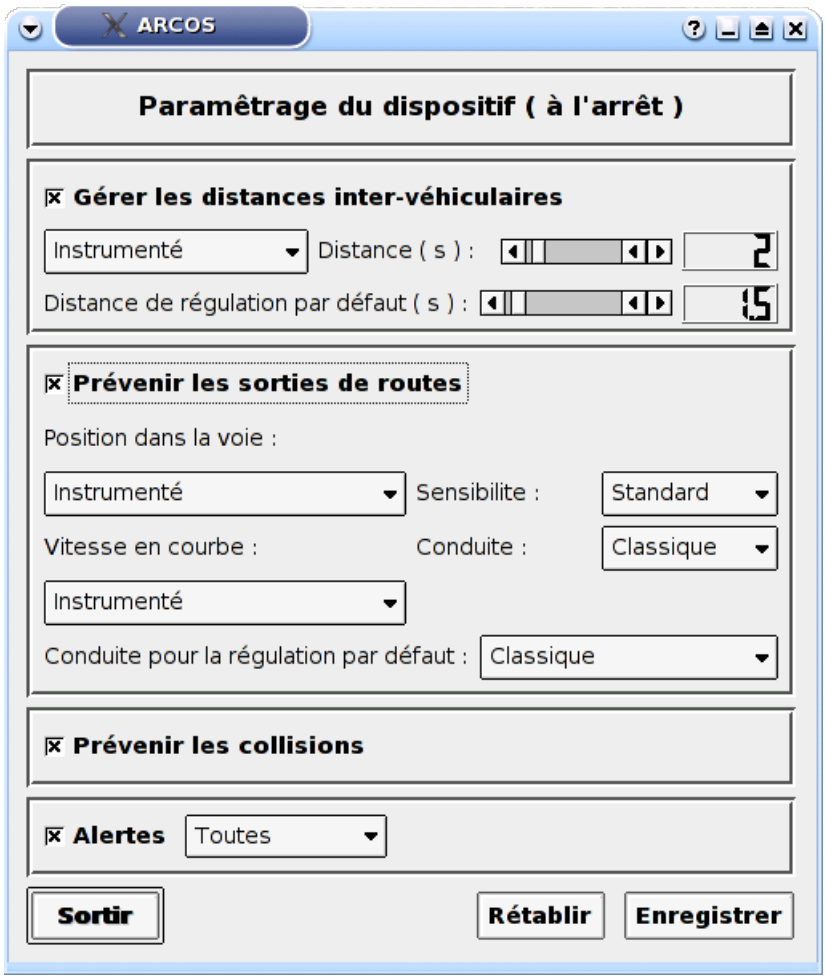
5.5.1 Fonction interdistance

–Connaître le zéro du gyromètre.
Pour cela, il existe deux moyens :
- par la console : canread_gyro
- en cliquant sur le bureau sur l'icône prévue à cet effet

Appuyer une fois sur Entrée lorsque le programme est lancé, et attendre plusieurs itérations de l'algorithme pour disposer de la valeur moyenne du gyromètre. Puis recopier cette valeur dans le fichier de données du DBITE, DonneesDBITE.txt

–Commencer par lancer l'interface QT
Pour cela, il existe deux moyens :
- par la console : IHM_ARCOSL2/IHM_ARCOSL
- en cliquant sur le bureau sur l'icône prévue à cet effet

–La fenêtre suivante est alors ouverte :



–Deux actions sont alors possibles :

- cliquer sur le bouton Rétablir si une configuration d'interdistance avait été au préalable enregistrée
- décocher toutes les fonctions sauf l'interdistance

–Choisir alors le mode d'avertissement :

- Instrumenté : seules apparaitront des alertes visuelles
- Avertissement : des alarmes sonores s'y ajoutent
- Suggestion d'action : on fait en plus vibrer la pédale d'accélérateur

–Régler la distance d'alerte pour les alarmes sonores (sur l'exemple, 2 s)

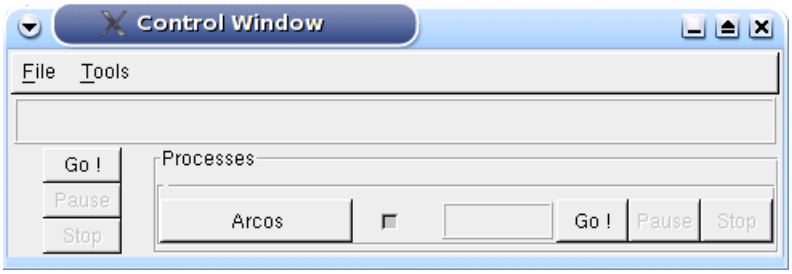
–Régler la distance d'alerte pour la vibration de la pédale (sur l'exemple, 1.5 s)

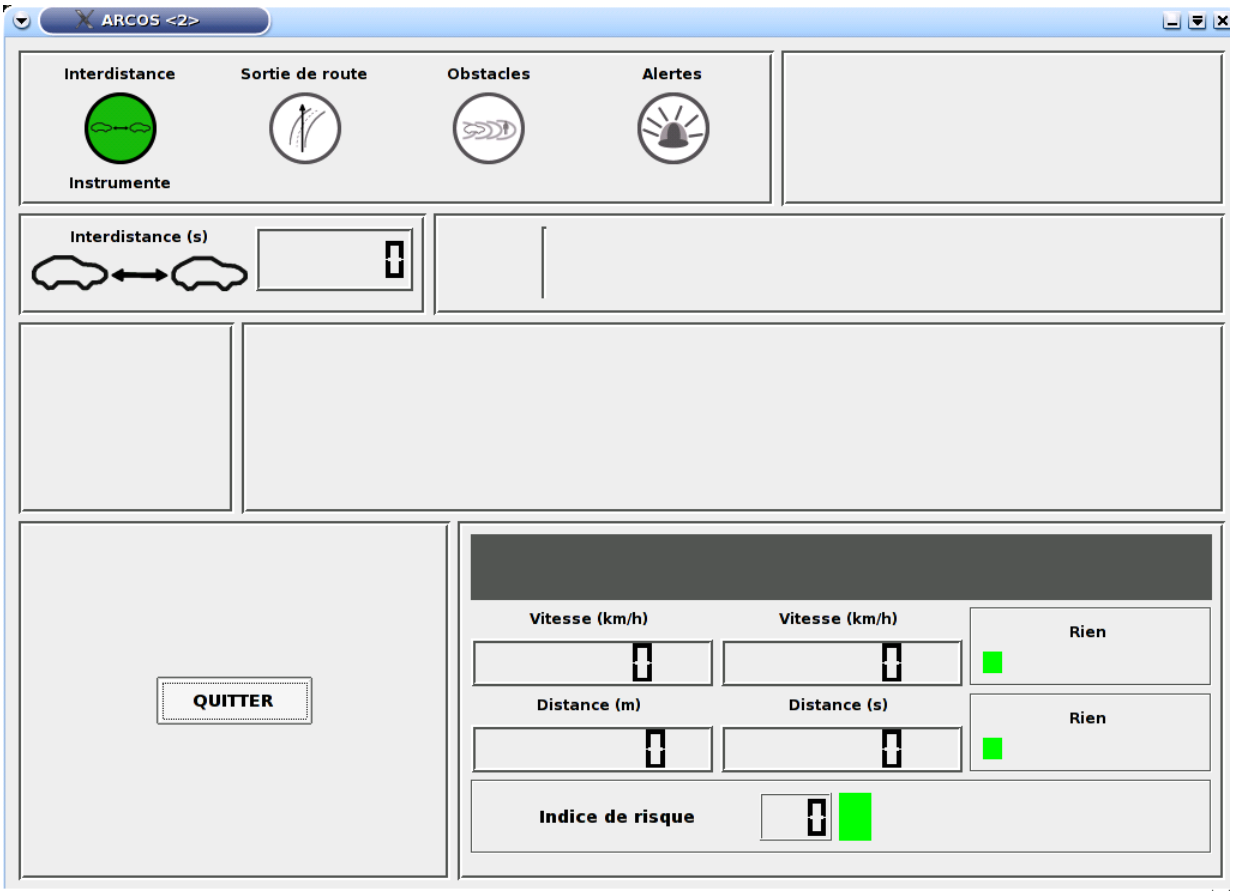
–Cliquer sur Enregistrer

Cette action a trois conséquences :

- Remplacer, dans le fichier de paramétrage du DBITE, “DonneesDBITE.txt”, les champs DISTANCE_ALERTE, DISTANCE_VIBRATION_PEDALE et MODE_INTERDISTANCE
- Lancer l'interface QT.
- Lancer le DBITE, via le script DistanceScript.bat contenu dans le répertoire IHM_ARCOSL2. Ce script appelle alors le ConfigFile Arcos/config_arcos_interdistance_vision, qui lui-même appelle le programme Arcos/pgm_arcos_interdistance_vision.

–Les fenêtres suivantes sont alors ouvertes :





–Cliquer sur Go !

La caméra est automatiquement configurée, et les algorithmes sont mis en route. Le viewer utilisé est celui combinant à la fois la vidéo et le télémètre. De plus, sur l'interface QT sont disponibles en bas à droite, la vitesse de notre véhicule, celle du véhicule que l'on suit (si tant est que l'on recoive cette information par modemradio), la distance en mètres et en secondes de la cible. Il y a également un indice de risque. Les différentes alarmes sont aussi relayées via cette interface QT.

–Pour arrêter, cliquer d'abord sur Stop dans la Control Window, puis sur Quitter dans l'interface QT

5.5.2 Fonction anticollision

–Connaître le zéro du gyromètre.

Pour cela, il existe deux moyens :

- par la console : canread_gyro
- en cliquant sur le bureau sur l'icône prévue à cet effet

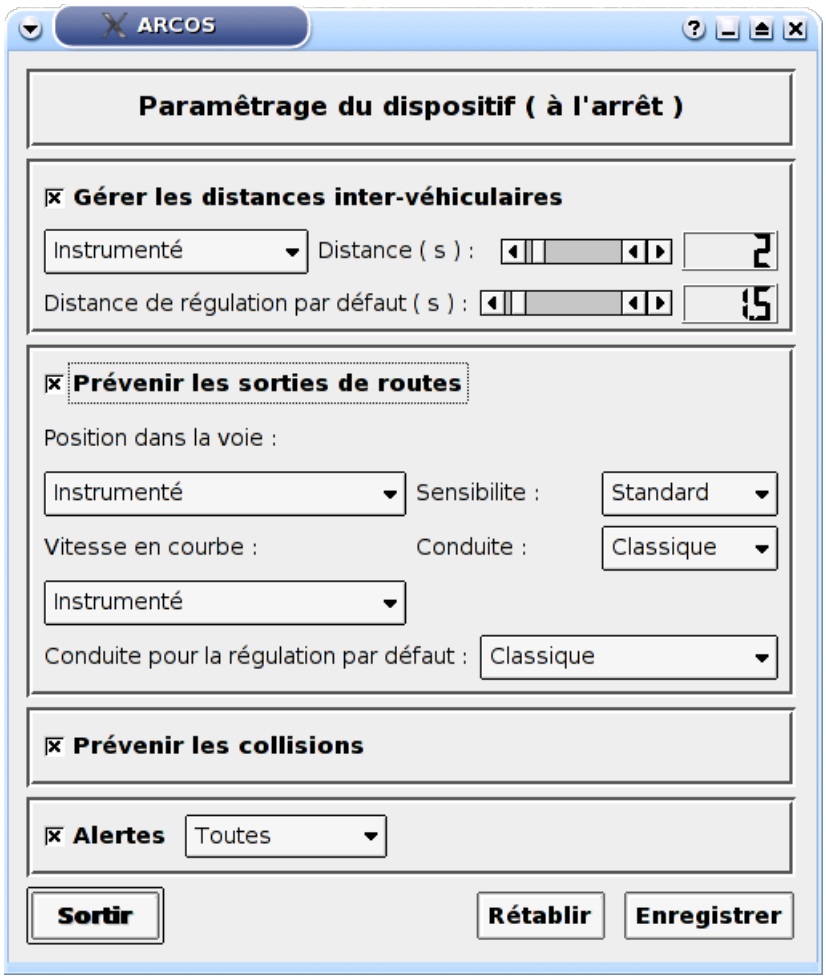
Appuyer une fois sur Entrée lorsque le programme est lancé, et attendre plusieurs itérations de l'algorithme pour disposer de la valeur moyenne du gyromètre. Puis recopier cette valeur dans le fichier de données du DBITE, DonneesDBITE.txt

–Commencer par lancer l'interface QT

Pour cela, il existe deux moyens :

- par la console : IHM_ARCOSL2/IHM_ARCOSL
- en cliquant sur le bureau sur l'icône prévue à cet effet

–La fenêtre suivante est alors ouverte :



–Deux actions sont alors possibles :

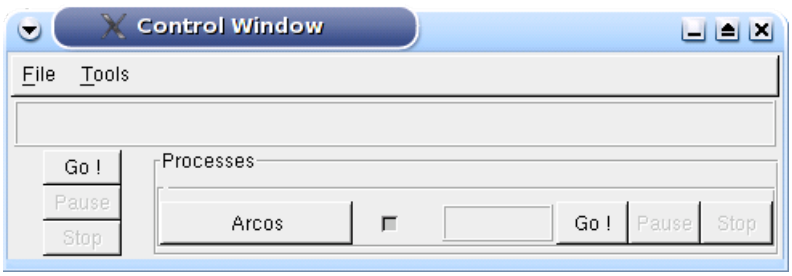
- cliquer sur le bouton Rétablir si une configuration d'anticollision avait été au préalable enregistrée
- décocher toutes les fonctions sauf l'anticollision

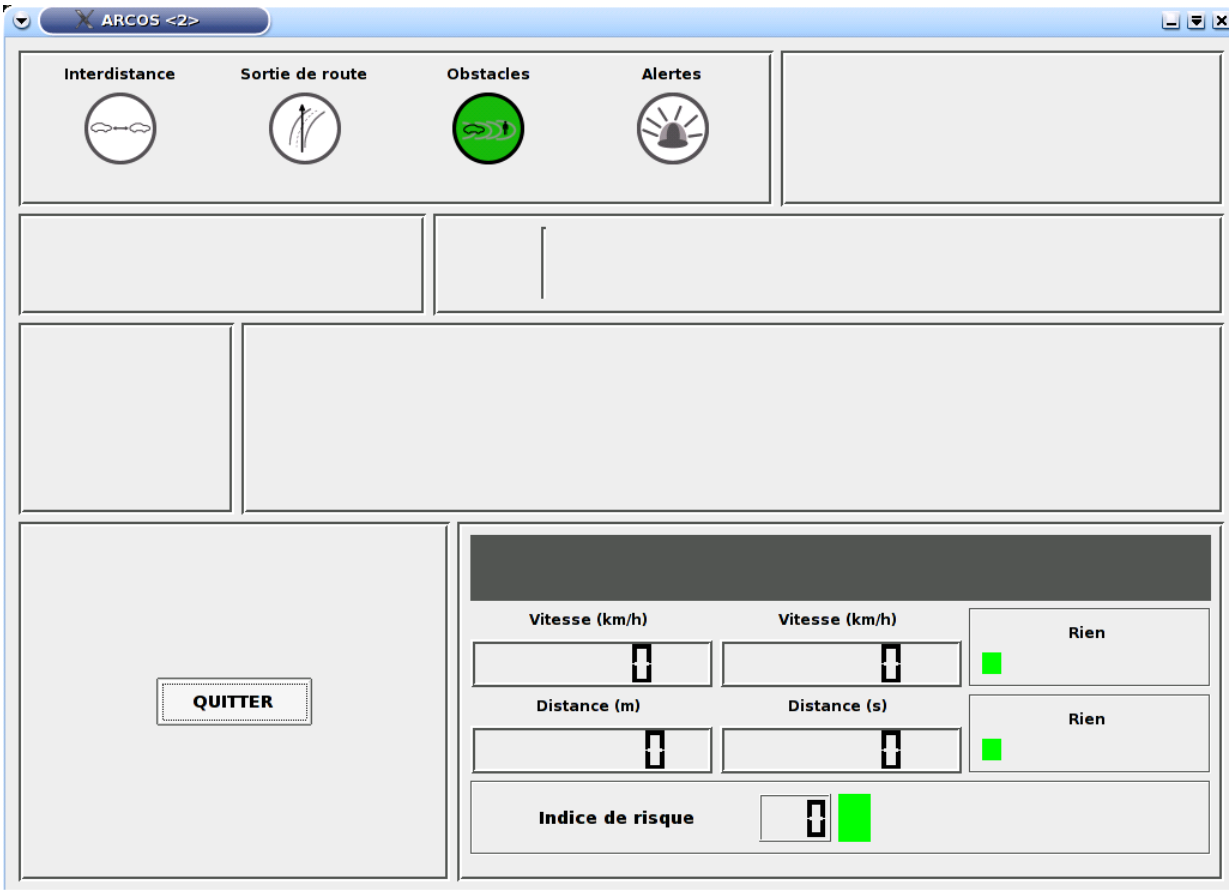
–Cliquer sur Enregistrer

Cette action a deuxconséquences :

- Lancer l'interface QT.
- Lancer le DBITE, via le script AnticollisionScript.bat contenu dans le répertoire IHM_ARCOSL2. Ce script appelle alors le ConfigFile Arcos/config_arcos_anticollision, qui lui-même appelle le programme Arcos/pgm_arcos_anticollision.

–Les fenêtres suivantes sont alors ouvertes :



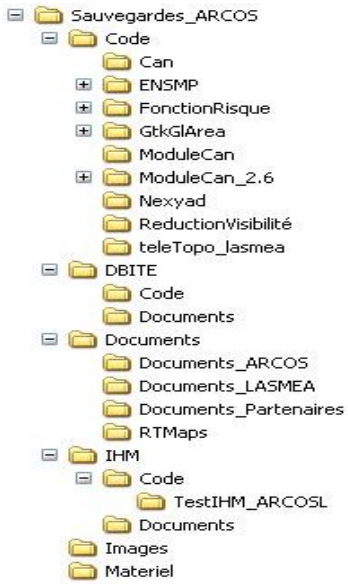


–Cliquer sur Go !

La caméra est automatiquement configurée, et les algorithmes sont mis en route. Le viewer utilisé est celui combinant à la fois la vidéo et le télémètre. De plus, sur l'interface QT est disponible en bas à droite la vitesse de notre véhicule. Il y a également un indice de risque. Les différentes alarmes sont aussi relayées via cette interface QT.

–Pour arrêter, cliquer d'abord sur Stop dans la Control Window, puis sur Quitter dans l'interface QT

5.6 Contenu du répertoire de sauvegardes d'ARCOS



On notera que l'on a laissé les codes informatiques utilisés lors de la journée du 28 octobre 2004 pour les démonstrations ARCOS, à savoir **IHM_QT_ARCOS** et **dbite_graph_arcos**. Cependant, ces codes ont été modifiés depuis. Le DBITE a été modifié de manière importante, au niveau de la documentation html, mais aussi au niveau du fichier contenant les données du DBITE. L'IHM a fait l'objet d'une retouche légère, mais essentielle pour fonctionner avec ce nouveau fichier de données. Il est donc essentiel de travailler dorénavant sur les archives **IHM_QT** et **dbite_graph_documente**.

Code informatique
- Concernant le CAN (read, write, gyro)
- Algorithme fourni par l'ENSMP, jamais intégré
- Fonction de risque fournie par le LIVIC
- Librairie GtkGlArea (Gtk et OpenGL)
- Module CAN pour un noyau 2.4
- Module CAN pour un noyau 2.6
- Algorithme de détection d'obstacles fourni par Nexyad
- Algorithmes de simulation de brouillard et pluie fournis par le LIVIC
- Utilisation du modemradio, fourni par le LIVIC

DBITE
- Différentes versions du DBITE (lire README)
- Documents de conception (dont celui-ci)

Documents relatifs à ARCOS
- Fournis par ARCOS
- Réalisés par le LASMEA
- Réalisés par d'autres partenaires
- Sur RTMaps

IHM
- Code IHM QT
  - Code de test de l'IHM
- Documents ARCOS sur l'IHM

Images

Documents sur le matériel embarqué dans Velac

6 Exploitation du Véhicule Expérimental du Lasmea pour l'Aide à la Conduite (VELAC)

6.1 Matériel embarqué dans VELAC

On commence par présenter brièvement le matériel embarqué à bord de VELAC pour introduire la façon dont il a été utilisé dans le cadre des fonctions ARCOS.

6.1.1 Télémètre laser

Le télémètre laser est un Riegel LMS- Z210-60. Il est intégré dans le pare-chocs de VELAC. Il permet de disposer d'une image 3D.

Ses caractéristiques sont les suivantes :

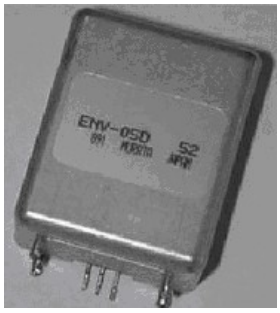
- télémètre 20 nappes
- angle de balayage horizontal : 60°
- angle de balayage vertical : 4.3°
- portée maximale : 100m
- cadence : environ 2 images par seconde
- image de taille 102x20

6.1.2 Caméra IEEE1394

La caméra utilisée est une caméra Sony DFW-VL500. Elle est fixée au niveau du rétroviseur intérieur de VELAC. Elle fournit environ une image toute les 40 ms.

6.1.3 Gyromètre

Le gyromètre utilisé est un gyromètre Gyrostar ENV05F-03. Il permet de mesurer la vitesse angulaire de VELAC. Il a été placé sous le siège arrière droit de VELAC, posé sur le sol.



6.1.4 Odomètre

L'odomètre utilisé envoie une trame sur le bus CAN chaque fois que VELAC parcourt 19.66 cm.

6.1.5 Bus CAN

La vitesse de transfert du CAN est de 250 Kbit/s. Les informations qui y circulent sont stockées dans des trames, chaque trame étant repérée par son identifiant. L'apparition des trames peut se faire de manière régulière ou évènementielle.

Les informations que l'on peut récupérer sur le bus CAN sont les suivantes :

- les trames du gyromètre
- les trames de l'odomètre

- signaux correspondant à l'appui des boutons au volant
- l'angle papillon
- pédales de frein et d'embrayage
- clignotants
- gps
- angle au volant

6.1.6 Modemradio

Le modemradio est un équipement qui permet la communication entre deux matériels distants par voie radio (environ 455MHz, modulation en fréquence et la fréquence centrale est sélectionnable) et de portée théorique de 2 km. Le mode de connexion avec un équipement se fait par voie série RS232 à la vitesse maximale de 9600 bauds. Les modèles employés dans ARCOS sont des Satelline 2ASxE de chez Satel. L'objectif recherché ici est de permettre de caractériser la distance estimée entre deux véhicule par la communication des position réelle des deux véhicules équipés de GPS précis.

6.1.7 Le PC

Les caractéristiques du pc embarqué sont les suivantes :

- processeur Intel Pentium 4 2.6 GHz
- mémoire 1024 DDR 400 MHz PC3200
- carte mère ASUS P4P800
- carte vidéo GF4 FX 5200 128MB
- carte IEEE1394 Texas Instruments
- carte CAN PCI EMS
- disques durs UDMA 133 40GB + UDMA 133 120GB

Le système d'exploitation utilisé est Linux avec la distribution Suse 9.1.

6.2 Fonctions ARCOS intégrées dans VELAC

6.2.1 Revue des activités

Le LASMEA intervient donc en tant qu'intégrateur pour deux fonctions ARCOS :

- Prévention des collisions
- Gestion des interdistances

L'intégration de ces fonctions a d'abord nécessité la mise en place d'une nouvelle version du DBITE (Driver Behaviour Interface Test Equipement), préalablement développé dans le projet européen Roadsense permettant une approche multicapteurs pour les entrées des algorithmes et des viewers. On se référera à la documentation concernant ce nouveau DBITE, « dbite_graph.doc », pour avoir davantage de détails.

Ensuite, des développements matériels et logiciels ont été effectués pour s'adapter aux besoins d'ARCOS, besoins définis au cours de réunions d'intégrateurs régulières. On cite entre autres :

- intégration de l'algorithme de détection de lignes blanches au DBITE.
- adaptation de l'algorithme de détection d'obstacles par télémètre : relaxation des

contraintes dans le cadre de la fonction Anticollision, ajout de champs pour décrire un obstacle, ...
- intégration de l'algorithme de tracking de véhicules par vision.
- développement de divers algorithmes, en particulier pour la gestion des données CAN, et de différents viewers pour le rendu scientifique.
- mise en place d'un gyromètre.
- mise en place d'un modemradio.
- vibration de la pédale d'accélérateur.

Outre les interfaces visuelles et haptiques (pédale d'accélérateur), on a également mis en place des interfaces sonores : deux alertes pour l'Anticollision et des bips variant en volume, fréquence et tonalité pour l'Interdistance. Voyons maintenant plus en détails ces deux fonctions.

6.2.2 Fonction Anticollision

La fonction Anticollision a pour objet la détection de tout type d'obstacle, pour peu qu'il soit suffisamment grand. Ceci va donc du piéton à la voiture, en passant par des cartons aux dimensions adéquates (on peut estimer une dimension minimum de 50cm x 80cm pour la face visible du carton, mais ceci mériterait d'être quantifié plus en détails).

6.2.2.1 Intégration de la fonction

La fonction utilise le matériel suivant :

- Le télémètre laser, utilisé pour la détection des obstacles. Les seuils de détection ont été grandement relaxés pour permettre une détection des petits obstacles, et en particulier des piétons. A noter aussi que, par le soft, la portée longitudinale (vers l'avant du véhicule) du télémètre a été limitée à 70 mètres, tandis que la portée latérale est restreinte à environ 4m.
- Le bus CAN et les trames suivantes en particulier :
  - Le gyromètre qui permet de déterminer la vitesse angulaire de notre véhicule.
  - L'odomètre : il envoie un top à chaque 19.66 cm parcouru.
- La caméra, utilisée uniquement pour l'interface. Peu importe donc la résolution employée, en revanche, le zoom doit être de 700 pour une projection correcte des obstacles sur la vidéo.

La détection d'obstacles se fait donc à partir des données télémétriques. Ensuite, il ne faut conserver que les obstacles se situant dans la zone de danger, définie comme étant la trajectoire empruntée par notre véhicule s'il gardait la même direction. On utilise donc pour cela le gyromètre.
Un obstacle est validé à partir de trois détections successives. Compte tenu de la fréquence du télémètre, ceci signifie une validation de l'obstacle 1.5 seconde après sa première détection. C'est pour cette raison que dans le cadre d'ARCOS, nous effectuions les tests à vitesse limitée (environ 20/25 km/h).

6.2.2.2 Résultats des scénarii

Présentons donc maintenant une synthèse des scénarii qui ont fait l'objet d'une évaluation lors des campagnes de mesure du 13 octobre 2004

a) Obstacle sur la trajectoire

Un obstacle immobile est placé au centre de la voie sur laquelle circule le véhicule. La cible est positionnée afin d'être visible par le véhicule de test au moins 3 secondes avant la collision éventuelle (soit 16.5 m à 20 km/h).

Pour ce scénario, les tests ont été concluants. Seul un test sur onze a été un demi-échec, un premier avertissement ayant été délivré au conducteur, mais pas le second. Ceci s'explique sans doute par une perte de l'obstacle au moment critique, phénomène relativement rare, mais qui ne peut jamais être exclu.

b) Objets de bords de voie

Un mannequin simulant un piéton est placé en bord de voie à 20 cm de l'autre coté de la ligne de marquage de la voie de circulation. Un véhicule immobile est placé sur la voie adjacente à celle sur laquelle circule le véhicule de test. Ce véhicule est placé à 20 cm de l'autre côté de la ligne de marquage du centre de la voie.

Lors de ce scénario, des fausses détections sont apparues pour quatre des onze tentatives. La première fois, ceci s'expliquait par le fait que l'on n'avait pas calibré le gyromètre, et donc la zone de danger était mal estimée. Les trois autres fausses détections ont permis de mettre en évidence un phénomène qui n'avait pas pu être défini jusqu'alors et que l'on peut présenter schématiquement comme suit : la route étant bordée par une haie, le télémètre détecte des obstacles dans cette haie, par exemple à 6 mètres sur le droite. A la prochaine image, il assimile cet obstacle à un nouvel obstacle détecté à 3 mètres sur la droite. Pour l'image suivante, une prévision de la position de cet obstacle est donc faite à 0 mètres sur la droite ! D'où une fausse détection. Ce phénomène a pu être corrigé en limitant la portée latérale du télémètre.

c) Obstacle sur la trajectoire et objet de bord de voie

Un véhicule immobile est placé sur la voie adjacente à celle sur laquelle circule le véhicule de test. Le véhicule immobile est placé à 20 cm de l'autre côté de la ligne de marquage du centre de la voie et reste en place pendant tout le déroulement du scénario. Un second obstacle immobile simulant un piéton est placé au centre de la voie sur laquelle circule le véhicule de test.

Ce scénario n'a révélé qu'une seule fausse alarme sur les dix tentatives, fausse alarme s'expliquant sans doute pour la même raison que celle exposée au-dessus.

d) Obstacle sur la trajectoire et objet de bord de voie

Un mannequin immobile simulant un piéton est placé en bord de voie à 20 cm de l'autre côté de la ligne de marquage de la voie de circulation. Un véhicule immobile est placé sur la voie adjacente à celle sur laquelle circule le véhicule de test. Le véhicule immobile est placé à 20 cm de l'autre côté de la ligne de marquage du centre de la voie. Un véhicule tiers parcourt la section rectiligne de la piste d'essai, au centre de la voie, à une vitesse de 20 km/h. Le conducteur du véhicule de test part derrière le véhicule tiers et stabilise sa distance de suivi à 2s (11 m), au moins 100 m avant le passage devant le premier objet.

Pour ce scénario, il y a eu quatre fausses détections sur les dix tests effectués. L'une d'entre elle semble imputable au phénomène déjà précédemment expliqué, les trois autres semblent concerner le véhicule tiers que l'on suit à une distance de onze mètres. Elles sont donc sans doute liées à une mauvaise estimation de la vitesse relative, et donc de la vitesse absolue du véhicule tiers.
On a remarqué que cette vitesse relative n'est pas toujours bien estimée. Pour tenter de résoudre ce problème pour les obstacles fixes, on a définit un seuil de 10 km/h. Si la vitesse absolue calculée est inférieure à ce seuil, alors on estime que l'obstacle est à l'arrêt et que sa vitesse absolue est donc 0 km/h. Dans ce cas, on peut donc dire que la vitesse relative de l'obstacle est égale à l'opposée de la vitesse absolue de notre véhicule.
On peut imaginer que dans le cas des fausses détections du scénario, la vitesse absolue du véhicule tiers a été par trois fois estimée comme étant inférieure à 10 km/h, donc il a été considéré comme étant fixe, et représentant un danger.

e) Piéton masqué traversant la voie

Un piéton initialement immobile est placé en bord de voie. Lorsque le véhicule de test se rapproche, le piéton se déplace perpendiculairement à la route, vers le centre de la voie, à la vitesse de 2 m/s. Le piéton apparaît sur la chaussée (avancé du piéton > 20 cm) lorsque le véhicule est à une distance de 16,5 m. Le piéton s'arrête au centre de la voie et fait demi-tour.

Sur dix essais, par trois fois le piéton n'a pas été détecté. Il faut signaler que ce piéton apparaît très brusquement. En conséquence de quoi aucun tracking ne peut être effectué avec le télémètre. Il faut attendre que le piéton soit à l'arrêt pour commencer les détections, puis attendre 1.5 secondes avant de valider l'obstacle. Ce délai très long explique les non-détections.

f) Obstacle démasqué par un véhicule qui déboîte

Cette variante a été jouée avec une silhouette en mousse simulant un piéton. Ce piéton en mousse immobile est placé au centre de la voie sur laquelle circule le véhicule de test. Un véhicule tiers parcourt la section rectiligne de la piste d'essai, au centre de la voie, à une vitesse de 20 km/h. Le conducteur du véhicule de test part derrière le véhicule tiers et stabilise sa distance de suivi à 2s (11 m), au moins 100 m avant le passage de l'obstacle. Le véhicule tiers déboîte soudainement et démasque l'obstacle immobile sur la voie.

Il y a eu deux non détections sonores sur les dix tests réalisés. Toutefois, l'obstacle a été signalé visuellement. Il semble donc que ces non détections soient liées à un non déclenchement du signal sonore, de manière inexpliquée.

6.2.3 Fonction Interdistance

La fonction Interdistance a pour objet la mesure de la distance séparant notre véhicule d'un véhicule le précédant. Son utilisation normale est routière ou autoroutière. La détection concerne uniquement les véhicules de type voitures ou camions.

6.2.3.1 Intégration de la fonction

La fonction utilise le matériel suivant :

- Le télémètre laser, utilisé pour la détection des véhicules. Les seuils de détection sont plus restrictifs que pour la fonction Anticollision, puisque nous ne détectons que des véhicules volumineux. La portée latérale du télémètre n'est plus restreinte à 4 mètres.

Elle est maintenant d'environ 6 mètres. En revanche, on limite toujours par le soft, la portée longitudinale du télémètre à 70 mètres.

- Le bus CAN et les trames suivantes en particulier :
  - oLe gyromètre qui permet de déterminer la vitesse angulaire de notre véhicule.
  - oL'odomètre : il envoie un top à chaque 19.66 cm parcouru.
  - oL'angle papillon, avec lequel on sait si la pédale d'accélérateur est enfoncée. La cas échéant, et si cela s'avère nécessaire, on fait vibrer la pédale d'accélérateur en envoyant une autre trame sur le bus CAN.
- La caméra, utilisée à la fois pour la détection des lignes blanches (positionnement de notre véhicule latéralement sur la chaussée), ainsi que pour le tracking d'obstacles par vision, afin de prendre le relais entre deux détections télémétriques. Sur ce dernier point, l'idée est évidemment de réagir plus vite par exemple lorsqu'un véhicule s'insère devant nous.
- Le modemradio, utilisé lors des campagnes de mesures du 13 octobre 2004. Son rôle était de fournir une réalité de terrain pour permettre d'apprécier les résultats de l'algorithme. Mais il n'est pas nécessaire pour le déroulement de la fonction.

La détection de véhicules se fait donc à la fois à partir des données télémétriques et des données vidéo. Ensuite, il ne faut conserver que les véhicules se situant dans la zone de danger, définie comme étant une approximation de la route, donnée à la fois par la détection des lignes blanches et par les données gyrométriques. Ceci ne donne qu'une estimation, car dans l'exemple où un virage se dresserait devant nous alors que nous sommes encore en ligne droite, nous ne pouvons pas prévoir la courbe.

6.2.3.2 Résultats des scénarii

Nous ne disposons pas de résultats permettant d'apprécier cette fonction, comme pour l'Anticollision. Des scénarii ont néanmoins été joués : variation de la distance séparant le véhicule suiveur du véhicule suivi par des paliers de vitesse, insertions et retraits de véhicules. Globalement, les résultats sont bons. Les alarmes sonores varient de manière cohérente, et les insertions et retraits sont bien gérés.

Sur ce dernier point, le temps de réaction est assez variable. Voici comment réagit théoriquement l'algorithme :

- le télémètre détecte un véhicule.
- le véhicule est validé après trois détection successives (environ 1.5 seconde).
- à partir du moment où un véhicule est validé, il est en plus suivi par vision entre deux données télémétriques.
- dès que ce véhicule s'insère sur notre voie, il est signalé.

Le tracking par vision devrait donc permettre une signalisation rapide du véhicule lors de son insertion. Néanmoins, ce tracking n'est possible que si le véhicule est visible entièrement dans l'image. En conséquence de quoi, si le véhicule s'insère trop proche de notre véhicule, le tracking par vision est inopérant, et la signalisation du véhicule est retardée par la faute de la lenteur du télémètre.

6.3 Collaborations mises en place dans le cadre d'ARCOS

6.3.1 Avec le LIVIC

6.3.1.1 Fonction de risque

Dans le cadre de la fonction Interdistances, il était prévu d'incorporer un indice de risque tenant compte de différents paramètres dont :

- Position du véhicule suiveur
- Vitesse du véhicule suiveur
- Accélération du véhicule suiveur
- Position du véhicule suivi
- Vitesse du véhicule suivi
- Distance de visibilité
- Capacité de freinage
- Hypothèses sur les types de conduite
- Masses des véhicules

Ce calcul de risque se fait par une fonction de risque qui nous a été fournie par le LIVIC. Il devait être utilisé lors des campagnes de mesures du 13 octobre 2004, mais tel n'a pas été le cas. Il n'a pas non plus été utilisé lors des démonstrations finales, car les données concernant le véhicule suivi (obtenues par modemradio, voir paragraphe suivant) n'étaient pas disponibles. Jamais donc cet indice de risque n'a vraiment pu être testé. Néanmoins, les résultats observés lors des tests effectués au LASMEA sont assez mitigés. Voici quelques raisons qui peuvent l'expliquer :

- Cette fonction de risque est difficile à calibrer. Il y a beaucoup de paramètres qui influent sur le calcul de l'indice.
- Elle était prévue pour fonctionner sous RTMaps, dans un contexte d'utilisation propre au LIVIC, avec différentes entrées qu'il était difficile de reproduire sur VELAC. On pense par exemple à la perte de visibilité dont la simulation sur VELAC se résume à une entrée Tout ou Rien.
- L'accélération de VELAC n'est pas une donnée fiable. Sa valeur est donnée par une dérivation de la vitesse, elle-même dérivée de la distance parcourue.

6.3.1.2 Utilisation d'un modem radio

La fonction de risque utilise les données issues de la réalité de terrain et transmises entre un véhicule du LIVIC et le VELAC, par modemradio. L'achat de ce modemradio a d'ailleurs été décidé pour être compatible avec le modèle qu'utilisait déjà le LIVIC. Ensuite, des échanges ont été mis en place pour définir la façon dont ces modemradios communiqueraient. Le LIVIC nous a fourni un soft de simulation d'envoi de trames pour que nous puissions tester le matériel au LASMEA. Un RoadIO dédié au modemradio pour une utilisation ARCOS a donc été implémentée dans le DBITE.

6.3.1.3 Simulation de brouillard et de pluie

Il était également prévu qu'une perte d'adhérence ou une perte de visibilité ait une influence sur le facteur de risque. S'il est difficile de simuler visuellement une perte d'adhérence, on peut en revanche donner à l'écran l'impression d'une visibilité réduite par un brouillard ou par la pluie. Dans cette optique, le LIVIC nous a fourni deux algorithmes de dégradation de l'image, que nous avons légèrement modifiés pour pouvoir les intégrer au DBITE. Ces algorithmes se trouvent dans le répertoire Algorithms/DegradationImage, et se nomment « brouillard » et « pluie ». L'idée est de leur fournir en entrée une image, et qu'ils retournent la même image dégradée. Pour ce faire, aussi bien pour la simulation de brouillard que de pluie, il y a un certain nombre de paramètres à fixer dans le code même de l'algorithme. Voici les résultats obtenus :

Le résultat pour la simulation de pluie est à peine perceptible ici. La raison est que cet algorithme est extrêmement lent, et pour tenter de limiter ceci, on simule une pluie très légère, donc avec un nombre de gouttes sur l'image très réduit. D'ailleurs, on observe ce même phénomène de lenteur avec la simulation de brouillard (en moins prononcé quand même). En conséquence de quoi, dans le cadre d'ARCOS, on a mis de côté cette simulation visuelle.

6.3.1.4 Interface QT

Une interface développée par le LIVIC afin de fournir un rendu commun à tous les véhicules de démonstration a également été intégrée dans le cadre d'ARCOS. Cette interface a un double rôle :

- Paramétrer le système : choix de la fonction à activer et caractéristiques de cette fonction (en particulier seuils de détection et mode d'alerte pour la fonction Interdistance).
- Apprécier visuellement les résultats par l'affichage de diverses informations à l'écran.

6.3.2 Avec l'ENSMP

Des échanges ont été initiés avec l'ENSMP pour le couplage de leur algorithme de détection de véhicules par vision et de notre algorithme de détection par télémètre. Dans cette optique, nous leur avons envoyé des données issues d'acquisitions réalisées au LASMEA. Comme l'ENSMP ne pouvait pas exploiter des données au format DBITE, nous avons intégré au DBITE des algorithmes de transformation de ces données.
L'idée est de fournir au final un seul fichier texte contenant les données datées pour la vitesse, le gyromètre, le télémètre et la vidéo. Pour cette dernière, elle est stockée sous forme de fichiers jpeg. L'ENSMP a pu appliquer son algorithme de détection de véhicules sur ces données.

En retour, nous avons eu le code informatique de cet algorithme, mais il est prévu pour fonctionner sous Windows, et en particulier avec la librairie IPL. Les différentes tentatives pour porter ce code sous Linux ont échoué.

6.3.3 Avec Nexyad

Une collaboration avec Nexyad a été mise en place pour intégrer leur algorithme de détection d'obstacles par vision au sein du DBITE. Pour des raisons de confidentialité, Nexyad ne nous a pas fourni le soft de l'algorithme, mais nous l'a transmis sous forme d'une librairie. Des échanges réguliers ont permis de définir les entrées et sorties de cette librairie pour qu'elle soit aisément exploitable par le LASMEA. De fait, son intégration dans le DBITE n'a pas posé de problèmes particuliers. En revanche, à l'utilisation, elle s'est révélée extrêmement lente, ce qu'a confirmé

Nexyad. Elle était donc inexploitable dans le cadre d'ARCOS. De plus, les différents tests réalisés n'ont pas permis d'apprécier la qualité de l'algorithme.

6.4 Diffusion de l'algorithme de détection des lignes blanches

Le LASMEA est également intervenu dans le cadre d'ARCOS pour diffuser auprès du LIVIC et de l'UTC l'algorithme de détection de lignes blanches qui a été développé au laboratoire.
Concernant l'UTC, les codes informatiques de l'algorithme ont été transmis dans leur intégralité pour la détection des lignes blanches, mais pas la partie concernant la localisation GPS.
Pour le LIVIC, la collaboration a demandé plus de travail, puisque le LIVIC souhaitait exploiter leur algorithme de détection des lignes, mais en bénéficiant de portions de l'algorithme du LASMEA, à savoir la définition de zones d'intérêt dans lesquelles effectuer la détection, le tracking des lignes, ainsi que l'estimation 3D. Conformément à ceci, le LASMEA n'a donc fourni que des bouts de code, adaptés aux besoins du LIVIC, lequel les a ensuite intégrés.
Il faut noter que ce programme ARCOS a permis de mettre en évidence les complémentarités dans les approches LASMEA et LIVIC et donné lieu à une collaboration au delà du cadre d'ARCOS. Des publications ont été par ailleurs issues de cette collaboration.

- Raphael LABAYRADE , Jerome DOURET, Jean LANEURIT, Roland CHAPUIS, « A Reliable and Robust Lane Detection System based on the Parallel Use of Three Algorithms for Driving Safety Assistance », Special Issue on Machine Vision and Applications EICE Transactions à paraître.

- J. Douret, R. Labeyrade, J. Laneurit, R. Chapuis « A Reliable and Robust Lane Detection System based on the Parallel Use of Three Algorithms for Driving Safety Assistance » IAPR Conference on Machine Vision Applications May 16-18, 2005.

7 Conclusion

De par son expertise réelle en matière d'expérimentation sur les Véhicules Intelligents, le LASMEA a eu en charge l'intégration de fonctions d'assistance à la conduite dans le cadre du programme ARCOS. Les fonctions « Anticollision » et « gestion des interdistances » ont ainsi été implantées sur le véhicule expérimental du LASMEA : VELAC.
Ce travail a nécessité d'une part l'intégration matérielle de divers capteurs (notamment capteur d'angle au volant et gyromètre) en plus de ceux déjà montés au sein du véhicule (odomètre, caméra, télémètre laser). D'autre part, une architecture logicielle a dû être mise en oeuvre pour assurer la gestion et le traitement en temps réel des diverses sources d'information. Cette architecture, fondée sur le système DBITE existant a été adaptée pour les besoins spécifiques aux fonctions. Par ailleurs, le rendu de résultats a impliqué la mise en place d'interface utilisateurs qu'il a fallu spécifier, implanter et valider.
En définitive, cette intégration, relativement lourde dans sa globalité a été réalisée dans son intégralité au LASMEA.
L'action spécifique du LASMEA concernant l'intégration de fonctions d'assistance à la conduite réalistes a ainsi contribué à faire d'ARCOS un succès. Les démonstrations du 28 octobre 2004, conclusions du projet, ont été la figure de proue de ce succès durant lesquelles VELAC, en particulier, a pu accueillir des dizaines de visiteurs à son bord. La démonstration de la fonction « Anticollision » mais surtout de la fonction « gestion des interdistances » pour laquelle la solution proposée par le LASMEA est extrêmement pertinente et fiable a contribué à la mise en valeur d'une faisabilité avancée de tels systèmes dans le véhicule de demain. Lors du carrefour à mi-parcours du PREDIT à Clermont-Ferrand au printemps 2005, les quatre véhicules de démonstrations ARCOS étaient en configuration d'essai. La démonstration à bord de VELAC a eu les honneurs de Monsieur de Ministre délégué à la recherche et de Monsieur le Président du Predit.

Fort de ces travaux et démonstrations, le LASMEA a noué divers contacts avec des partenaires tant industriels pour des actions prospectives (par exemple l'entreprise Prosign pour des actions de marquage automatique de bandes au sol) que d'autres laboratoires (notamment le LIVIC avec lequel des actions de collaboration ont été entreprises, ce en partenariat avec le constructeur RENAULT).