Projet de département (IMI)

Extraction de coordonnées de véhicules à partir d'une scène vidéo

Renault

Alicia Quilez
Aurélien Boffy
Flavien Billard
Thomas De Soza

Tuteurs
Loïc Joly et Nicolas Wagner

22 juillet 2005

Table des matières

1 Introduction **2**
- 1.1 Description du projet . . . 2
  - 1.1.1 Renault . . . 2
  - 1.1.2 Le logiciel de simulation *SCANeR* . . . 2
  - 1.1.3 Le sujet expliqué . . . 3
- 1.2 Organisation . . . 3
  - 1.2.1 Les rencontres . . . 3
  - 1.2.2 La méthodologie adoptée . . . 4

2 Algorithmes **6**
- 2.1 Algorithme reposant sur le BMA . . . 6
  - 2.1.1 L'algorithme de *Block Matching* (BMA) . . . 6
  - 2.1.2 Le lissage du champ de vecteurs . . . 8
  - 2.1.3 L'attribution des labels . . . 8
  - 2.1.4 Le suivi des véhicules . . . 8
- 2.2 Soustraction de fond & Appariement de blocs . . . 9
  - 2.2.1 Détection des véhicules . . . 10
  - 2.2.2 Suivi des véhicules . . . 11
- 2.3 Extraction de coordonnées . . . 13
  - 2.3.1 Un problème à part entière . . . 13
  - 2.3.2 La modélisation de la caméra . . . 14

3 Tests et résultats **17**
- 3.1 Les vidéos . . . 17
- 3.2 Algorithme reposant sur le BMA . . . 18
  - 3.2.1 Vidéo de l'autoroute . . . 18
  - 3.2.2 Vidéo de Karlsruhe . . . 19
- 3.3 Soustraction de fond & Appariement de blocs . . . 19
  - 3.3.1 La détection de véhicules . . . 19
  - 3.3.2 Le suivi des véhicules . . . 20
- 3.4 L'extraction en pratique . . . 21

Bilan **22**

Bibliographie **23**

Chapitre 1

Introduction

Ce chapitre a un double objectif. Après avoir présenté le projet, il s'agira d'une part d'expliquer clairement le sujet pour le lecteur non familier avec les concepts de la vision et du traitement d'images par ordinateur, et d'autre part de présenter l'organisation au sein de notre équipe ainsi que la méthodologie choisie pour réaliser le projet une fois les problématiques identifiées.

1.1 Description du projet

1.1.1 Renault

Créée en 1898, l'entreprise Renault est maintenant un groupe présent dans plus de quarante pays dont l'activité se divise en deux branches : la branche automobile et la branche financement des ventes.
Au sein de la première, le Technocentre situé à Guyancourt regroupe depuis 1998 la majeure partie des équipes de Recherche et Développement, 11000 ingénieurs et techniciens y travaillent. Il comprend notamment le Centre Technique de Simulation qui développe et teste des outils liés à la réalité virtuelle et au "véhicule numérique".

Renault utilise des simulateurs de conduite depuis une dizaine d'années : un nouveau simulateur dynamique a été installé au Technocentre en 1999, et le projet européen *Ultimate* développé en partenariat avec le CNRS vise à concevoir un simulateur de conduite à haute performance dont les applications concernent la conception innovante de voitures dans le domaine des véhicules dynamiques et l'étude de la sécurité routière liée au comportement du conducteur.

1.1.2 Le logiciel de simulation *SCANeR*

Le simulateur permet ainsi de tester en amont les réactions des usagers face à un nouveau modèle de voiture. En effet, construire un concept-car ou un prototype coûte très cher et il est difficile de réaliser des tests sur routes avec trafic : il n'est pas toujours possible de le faire dans la circulation publique pour des raisons de sécurité. Le simulateur recrée avec précision toutes les situations qu'on peut rencontrer dans la circulation de tous les jours.

Une des clés du projet *Ultimate* est le logiciel de simulation SCANeR qui, entre autre, crée et gère le trafic environnant le véhicule plongé dans la circulation. C'est sur un composant de cette pièce maîtresse que porte le travail du stagiaire long de l'ENPC Nicolas Wagner qui a proposé, avec son tuteur M. Joly, ce projet de département IMI.

1.1.3 Le sujet expliqué

Une fois un modèle défini pour la gestion du trafic dans le logiciel de simulation, il est nécessaire de calibrer ce modèle afin que le résultat soit le plus proche possible des situations qu'on rencontre dans la vie quotidienne. Ce processus de calibration était jusqu'ici réalisé "à la main" (c'est à dire de manière subjective), ce qui conduit notamment à une simulation d'un trafic peu réaliste.
Le projet qui nous a été confié se propose d'automatiser ce processus à partir de vidéos filmant une situation de trafic routier donnée. Il s'agit d'extraire les coordonnées des véhicules afin d'obtenir certaines caractéristiques de la circulation permettant de définir les stratégies qu'adoptent les conducteurs : l'écart temporel avec le véhicule de devant, le comportement lors du dépassement, ...

A partir d'une vidéo d'une scène de trafic, nous devions reproduire une carte donnant en vue de dessus (comme si la scène était filmée à la verticale) l'évolution du trafic en représentant par exemple les voitures par des rectangles colorés, et en reconstituant la trajectoire de chacun des véhicules sur la scène.

Ces premières explications permettent de comprendre à quel point les restrictions en matière de type de vidéos utilisées (caméra de surveillance, caméra placée sur un mât, un pont, ...) et de type de terrain filmé (plat, vallonné, en courbe, ...) jouent un rôle important. Cependant, nous ne disposions d'aucun renseignement supplémentaire avant de commencer à travailler sur le projet. Un travail préliminaire consistait donc à prendre de plus amples informations auprès des commanditaires que nous avons rencontrés au Technocentre de Renault de Guyancourt, puis à organiser notre travail.

1.2 Organisation

1.2.1 Les rencontres

Une visite au Technocentre

Nous avons commencé par rencontrer au Technocentre de Renault Nicolas Wagner, stagiaire long de l'ENPC, et son tuteur M. Joly, qui ont proposé ce projet. Ce fut l'occasion pour nous de solliciter plus d'informations tant sur leurs attentes, que sur l'utilisation possible de notre travail.

Nous avons pu recueillir de précieuses informations sur le type de vidéos à utiliser pour notre travail : M. Joly exigeait en effet de pouvoir travailler sur des scènes qui pouvaient être planes, mais la route devait pouvoir ne pas être rectiligne (carrefours par exemple).

Accessoirement le logiciel que nous serions amenés à développer pour ce projet devait être open-source afin d'éviter tout contentieux lors de la remise du code. Nicolas Wagner nous a d'ailleurs donné à cette occasion quelques pistes sur les outils à utiliser pour programmer.

Un spécialiste du *tracking*

Nous avions tous choisi de travailler sur ce projet car il nous permettrait d'approfondir nos connaissances en vision par ordinateur et traitement d'images mais aussi en programmation.

Dans les divers domaines de la vision par ordinateur, il s'est vite avéré que nous aurions besoin de mettre en oeuvre des techniques de *tracking* afin de suivre des objets, ici des voitures, au cours de leur mouvement dans la vidéo. Le *tracking* est un domaine en pleine expansion, car ses applications sont de plus en plus nombreuses dans un monde où tout tend à être automatisé.

Nous avions un peu abordé ce sujet en cours avec notre professeur de vision par ordinateur M. Paragios, qui se révéla être un spécialiste de cette technique. C'est pourquoi, afin de ne pas nous égarer dès le début, nous sommes allés lui demander quelques renseignements. M. Paragios a été très aimable et a répondu à toutes nos questions. Il nous a permis de faire le lien entre nos enseignements à l'école et notre projet pour une entreprise.

1.2.2 La méthodologie adoptée

travail préliminaire

Avant de nous lancer dans l'écriture des programmes, nous avons décidé de filmer notre propre situation de trafic afin d'en maîtriser la composition (cf. section 3.1).

Nous avons ensuite travaillé pendant quelques séances à plusieurs afin de nous former à l'utilisation de la bibliothèque graphique que nous avions décidé d'utiliser. Le logiciel devant être open-source, nous avons eu quelques difficultés à trouver la bonne bibliothèque. Renseignements pris auprès de M. Keriven, développeur de la Winlib, bibliothèque utilisée pour les cours d'informatique de l'ENPC, nous avons finalement choisi d'utiliser la bibliothèque open-source (contrairement à la Winlib) OpenCV développée par Intel. L'intégration d'OpenCV à l'environnement Visual C++ et la création de projets nous a notamment posé beaucoup de difficultés.

Implémentation informatique

Après avoir rassemblé nos différentes idées d'algorithmes et plusieurs articles que nous avons cherchés en suivant les indications de M. Paragios, nous avons décidé d'implémenter la méthode décrite dans la section 2.1. Pour ce faire, nous nous sommes le plus souvent séparés en 2 groupes de 2 afin d'être plus productifs. Après quelques semaines, les résultats n'étant pas aussi satisfaisants que nous l'aurions souhaité, nous sommes retournés voir M. Paragios qui nous a donné de nouvelles pistes.

Nous étions alors en face d'un dilemme : nous avions le choix entre continuer l'implémentation de la première méthode pour essayer de l'améliorer ou programmer un nouvel algorithme qui tirerait partie des nouvelles indications dont nous disposions. Nous avons décidé de faire les deux en parallèle.

Fonctionnement interne

Pour éviter qu'un des deux binômes ne soit trop déconnecté du travail de l'autre, nous avons pris soin d'effectuer des réunions hebdomadaires.

Il s'est vite avéré que pour l'échange d'informations, qu'il s'agisse des articles décrivant les algorithmes, de l'installation de la bibliothèque ou des sources des programmes, nous aurions besoin d'un espace. C'est dans cette optique que nous avons créé un site web où nous pouvions déposer nos fichiers, afin que l'équipe puisse les consulter librement et pour que nos tuteurs puissent constater l'avancement du projet[1].

Au final, une fois les deux algorithmes de *tracking* terminés, nous nous sommes tous penchés sur la question de l'extraction de coordonnées.

[1] http ://projet-renault.enpc.org

Chapitre 2

Algorithmes

Dans cette partie, nous présentons les différents algorithmes que nous avons implémentés. Plus précisément, deux algorithmes de *tracking* sont expliqués ainsi que les méthodes d'extraction de coordonnées en vision par ordinateur.

2.1 Algorithme reposant sur le BMA

La première méthode que nous avons mise en oeuvre s'appuie sur un article publié par le laboratoire d'informatique de l'université de Bologne [1]. L'algorithme dit de *Block Matching*, ou mise en correspondance de blocs, joue un rôle central dans cette approche car il fournit une estimation du mouvement qui va nous permettre de suivre les véhicules puis d'en récupérer les coordonnées.

2.1.1 L'algorithme de *Block Matching* (BMA)

Principe Cet algorithme s'inspire des techniques mises en oeuvre en compression vidéo et est notamment utilisé dans le format de compression MPEG. Il permet en effet de ne stocker dans la vidéo compressée qu'un certain nombre d'images (une par seconde par exemple) et de reconstituer les autres à partir des différences par rapport aux images précédentes. Le principe général du *block matching* est en effet d'exploiter les redondances temporelles existant entre des images consécutives.

On considère donc deux *frames* successives qu'on partage en blocs de taille fixe. Supposons que nos images soient de taille $w \times h$ où $w = 720\ px$ et $h = 576\ px$. On les découpe par exemple en blocs de 8 pixels par 8 pixels soit 90 sur 72 blocs.

On considère chaque bloc de la seconde image qu'on déplace dans une fenêtre dite de recherche pour trouver dans la première image le bloc de même taille qui lui ressemble le plus. On effectue cette recherche dans une fenêtre autour du bloc ciblé, afin de limiter le nombre de comparaisons. La position choisie dans la première image détermine un vecteur de déplacement dont on stocke les coordonnées : on associe donc à chaque image un champ de vecteurs de déplacements. On a ainsi une idée du déplacement pour un petit bloc de taille 8*8 dans un temps qui est *l'interframe*, soit 1/25ème de seconde. Idéalement, il

ne faudrait qu'un seul vecteur déplacement par voiture, c'est pourquoi les étapes suivantes de l'algorithme s'attacheront à regrouper les blocs qui appartiennent à un même véhicule.

Les critères de ressemblance Il existe plusieurs critères de ressemblance pour comparer deux blocs. La vraie quantité statistique est la corrélation mais les calculs sont plus complexes et plus longs pour un résultat qui n'est pas toujours meilleur qu'avec des critères plus simples. D'autres critères ont donc été essayés, notamment la corrélation croisée normalisée, et la comparaison des histogrammes des deux blocs (c'est à dire des distributions des niveaux de gris). Dans ce cas, l'information spatiale est perdue donc il est plus difficile d'identifier les textures.

Nos programmes, eux, minimisent la somme, sur le bloc, des carrés des différences d'intensité pixel par pixel :

$$m_1 = \underset{m}{\text{Argmin}} \sum_{i,j} \Big(I_1\big(m + (i,j)\big) - I_2\big(m_2 + (i,j)\big) \Big)^2$$

Dans cette équation, I_1 est par exemple l'image courante et I_2 l'image au temps $t + 1$. i et j parcourent les blocs en hauteur et en largeur. $I_k(i,j)$ est l'intensité du pixel (i,j) de l'image k. m_2 et m sont les origines des deux blocs. m_2 est fixé et on cherche m_1 qui sera l'origine du bloc dans l'image précédente.

Les paramètres La taille de la zone de recherche influe grandement sur la vitesse d'exécution de l'algorithme et doit être choisie en fonction de la vitesse des voitures apparaissant à l'écran et en fonction de la taille des blocs.

Pour éviter qu'il y ait trop de bruit sur le champ de vecteurs et gagner en temps de calcul, on peut définir un seuil haut de corrélation et toujours commencer par regarder si chaque bloc corrèle avec lui-même dans l'image précédente à un niveau supérieur à ce seuil. Si c'est le cas, on valide un déplacement nul de ce bloc. Ainsi, comme la plus grande partie de l'image est statique, cette astuce permet d'accélérer significativement l'exécution.

La taille des blocs est en général fixée à 8×8 ou 16×16 pixels : on doit adapter cette taille à la taille caractéristique des voitures qui apparaissent dans la vidéo. Si les blocs sont trop petits, on a du mal à obtenir des champs de vecteurs homogènes car on ne distingue plus de texture, et si les blocs sont trop gros, on ne recouvre jamais correctement une voiture car la grille de blocs est fixe et il est rare qu'une voiture s'inscrive parfaitement dans un regroupement de gros blocs. Un gros point noir de cette méthode réside donc dans le choix de la taille de ces blocs.

Une amélioration On peut améliorer la précision du découpage en blocs mais aussi la vitesse de l'algorithme en utilisant l'*Adaptive Block Matching*[2]. Ici la taille des blocs est variable au sens où l'on définit une taille de blocs maximum, par exemple 32×32 pixels, et une taille minimum, par exemple 8×8 pixels. On commence par lancer le BMA avec les gros blocs et si, pour

un bloc, le vecteur déplacement trouvé est non nul, alors on relance le BMA sur ce bloc mais avec une taille de bloc deux fois plus petite. Si en revanche le vecteur déplacement était nul, alors on suppose que tous les blocs plus petits qu'il contient ont un vecteur déplacement nul (ce qui permet donc aussi un gain de temps dans l'exécution de l'algorithme).

2.1.2 Le lissage du champ de vecteurs

Malgré les différentes précautions évoquées, le champ de vecteurs obtenu par l'algorithme de *Block Matching* est en général bruité et inhomogène, ce qui peut entraîner une sur-segmentation des véhicules : plus tard lorsqu'on tentera de regrouper les blocs, on risque de découper une seule voiture en plusieurs morceaux distincts.

Pour y remédier on utilise un filtre médian qui régularise le champ des vecteurs déplacement. Ce filtre remplace chaque vecteur par le vecteur médian de l'ensemble constitué par le vecteur lui-même et par ses huit plus proches voisins (le vecteur médian étant défini comme celui qui minimise la somme des distances au carré de tous les autres vecteurs à lui-même). Ce procédé permet non seulement d'éliminer un peu le bruit, mais surtout de rendre au sein d'un même futur objet le champ de vecteurs à peu près homogène.

On ne remplace que les vecteurs non nuls pour éviter l'apparition de bruit. De plus, pour le calcul du vecteur médian, on ne considère que les voisins non nuls dans le calcul des distances, ceci afin d'éviter de transformer les vecteurs correspondant aux bords d'un objet en vecteurs nuls.

2.1.3 L'attribution des labels

Maintenant que le champ de vecteurs est plus régulier, on regroupe les blocs dont les vecteurs de déplacements sont similaires. Pour ce faire, les blocs appartenant à un même véhicule se voient attribuer un numéro identique, ou label. On donne une étiquette à chaque objet tout en conservant la structure d'image découpée en bloc.

Pour ce faire, on parcourt l'ensemble des blocs, et chaque bloc non étiqueté devient une "graine" : on lui attribue un nouveau label, et on propage de manière récursive ce nouveau label aux blocs qui l'entourent et dont les vecteurs de déplacements sont suffisamment proches du vecteur de notre "graine" (i.e. la norme de la différence entre les deux vecteurs ne dépasse pas un certain seuil).

A la fin de cette étape, on dispose d'une matrice qui associe un label à chaque bloc. On crée alors une nouvelle matrice qui associe cette fois un label à chaque pixel afin de pouvoir, dans la dernière étape, estimer des vecteurs déplacement pour chaque objet étiqueté dans la scène.

2.1.4 Le suivi des véhicules

Idée On est désormais capable de discerner les différents véhicules sur chaque image. Il faut alors faire en sorte que chaque label désigne le même véhicule

d'une image à l'autre. En d'autres termes il faut suivre un objet étiqueté depuis son apparition dans la scène jusqu'à sa disparition.

Supposons donc qu'à l'instant t, on ait suivi avec succès un objet étiqueté avec le label A. On se place à l'instant $t + 1$: les différentes étapes jusqu'à l'attribution des labels ont été effectuées, notre objet s'est déplacé, a légèrement changé de forme et est étiqueté avec le label D. Le but va être de retrouver ce label D dans l'image précédente, c'est-à-dire de lui attribuer le label A.

Principe On commence donc par estimer le vecteur déplacement moyen de chaque objet ayant un label temporaire en prenant naturellement la moyenne de chaque vecteur déplacement des petits blocs le constituant.

En inversant ce vecteur de déplacement moyen, on peut retrouver la position supposée de l'objet sur l'image précédente. En superposant cette ancienne position avec la matrice des labels correspondant à l'image précédente, on détermine le label qui présente le recouvrement le plus important avec notre objet temporaire et on le lui affecte. S'il n'existe aucun recouvrement, c'est qu'un véhicule est apparu et on lui affecte un nouveau label. Réciproquement si un objet de l'image précédente ne correspond à aucun objet de l'image courante alors celui-ci a disparu. On peut ainsi tenir un tableau des véhicules actuellement suivis.

Finalement on dispose donc d'une méthode de suivi d'objets, dont on présente les performances en section 3.2.

Remarque On pourrait se demander pourquoi découper l'image en petits blocs pour ensuite les regrouper. Pourquoi ne pas tout de suite utiliser des blocs de la taille caractéristique des voitures ? Il faut bien comprendre que ce n'est pas possible en laissant l'algorithme en état puisque la grille de blocs étant fixe il n'est pas possible qu'une voiture soit recouverte parfaitement par un bloc aussi gros soit-il.

Cependant l'idée de choisir des blocs de la taille des voitures peut s'avérer bonne car, comme on l'a fait remarquer, plus un bloc est gros, plus il est texturé et donc plus il est facile de le faire correspondre précisément à un autre. Il est donc nécessaire de s'affranchir d'un découpage en blocs et d'imaginer une technique différente de *tracking*. C'est l'objet de la seconde méthode que nous avons implémentée.

2.2 Soustraction de fond & Appariement de blocs

Nous allons distinguer deux principales étapes dans cet algorithme. Lors de l'entrée d'un véhicule sur la route, un nouveau bloc doit être créé et positionné sur la voiture entrant dans la scène (détection du véhicule). Ce bloc va ensuite suivre le véhicule tout au long de son parcours jusqu'à ce qu'il sorte de l'écran (suivi du véhicule).

2.2.1 Détection des véhicules

Principe La première étape consiste à détecter l'entrée d'un véhicule. Pour ceci, nous examinons une petite zone à l'entrée de chaque voie dans laquelle nous regardons si quelque chose (normalement une voiture) bouge : c'est la **zone de détection**, représentée par un rectangle bleu plein sur la figure 2.1.
Si c'est le cas, on attend que l'objet sorte de la zone et on peut à ce moment savoir à peu près où il se situe. Pour chaque voie, nous avons en fait défini une seconde zone où la voiture est censée se trouver après avoir quitté la zone de détection et dans laquelle nous cherchons à approcher au mieux sa position : c'est la **zone de recherche du véhicule détecté** représentée par un rectangle bleu transparent sur la figure. Nous sommes désormais capables de l'encadrer par un bloc (représenté par un bloc rouge dans la figure 2.2) qui la suivra jusqu'à ce qu'elle sorte de la zone filmée.

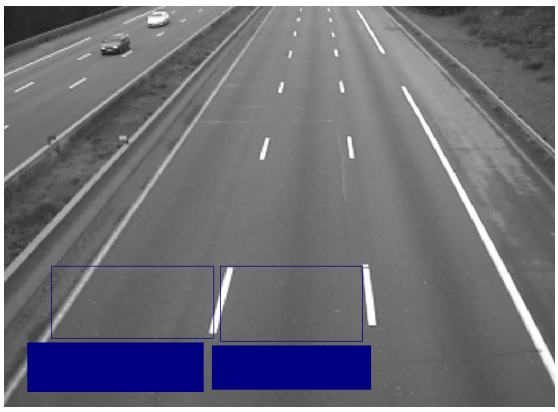

FIG. 2.1 – La zone de détection et la zone de recherche d'un véhicule entrant

FIG. 2.2 – Après la détection du véhicule, un bloc est créé et placé sur le véhicule

Soustraction de fond Le procédé qui est à la base de cette détection de véhicule est la soustraction de fond, ou soustraction d'arrière-plan (*Background Subtraction* [3]). Il consiste à construire un modèle de l'arrière-plan de la vidéo puis de soustraire l'image courante à ce fond pour idéalement ne conserver que les objets qui ne sont pas partie intégrante de l'arrière-plan, à savoir les véhicules qui se déplacent, mais aussi les piétons et autres éléments mobiles.

Il existe différentes techniques de soustraction de fond plus ou moins complexes et nous avons décidé d'implémenter une méthode assez basique car nous travaillons le plus souvent avec des vidéos "idéales", en ce sens que le fond est généralement statique (lorsque l'arrière-plan est une scène naturelle avec des arbres dont les branches bougent ou des nuages qui se déplacent, il est plus difficile de distinguer le fond des éléments dynamiques) et la durée de la vidéo assez courte, ce qui évite d'éventuels changements importants de luminosité. Ainsi, soit nous utilisons une image de la vidéo où il n'y a aucune voiture comme arrière-plan, soit nous considérons la moyenne des N images précédant l'image courante.

En soustrayant l'arrière-plan de l'image courante, nous obtenons une nouvelle image où la plupart des pixels ont une intensité proche de 0. Les pixels dont le niveau de gris est supérieur à un seuil fixé sont considérés comme appartenant à un élément mobile.

Initialisation Ainsi, dans la **zone de détection** destinée à repérer les véhicules entrant dans la scène, nous examinons la proportion de pixels "mobiles". Lorsque celle-ci dépasse un certain seuil haut, nous considérons qu'un véhicule est présent dans le cadre, et lorsqu'elle passe sous un seuil bas, c'est que la voiture est sortie de la fenêtre de détection. Nous parcourons alors la **zone de recherche du véhicule détecté**, où est censé se trouver approximativement le véhicule. Dans cette fenêtre de recherche, nous cherchons la position précise du véhicule qui correspond à la zone où se situent le plus de pixels "mobiles". Un bloc entourant une partie du véhicule étant désormais placé (bloc rouge sur la figure 2.2), la seconde phase, qui correspond au *tracking* à proprement parler, peut débuter.

2.2.2 Suivi des véhicules

Principe Pour suivre les voitures, nous utilisons aussi une méthode par appariement de blocs (*block matching*) : un bloc étant placé sur le véhicule à l'instant t sur l'image I_t, nous cherchons à le repositionner au temps $t+1$ sur la même partie de la voiture : nous recherchons dans I_{t+1} le bloc dont le contenu ressemble le plus au contenu du bloc sur I_t. Nous utilisons comme précédemment la minimisation de la somme des carrés des différences d'intensité pixel par pixel comme critère de ressemblance pour comparer 2 blocs.

Fenêtre de recherche On ne parcourt pas l'intégralité de l'image pour trouver le bloc car une telle recherche exhaustive serait beaucoup trop longue. On parcourt en fait tous les blocs situés à l'intérieur d'une zone déterminée appelée fenêtre de recherche et représentée en vert sur la figure 2.2.

L'idée est d'utiliser la connaissance que l'on a de la situation. On sait que le véhicule ne peut traverser la scène entre deux images successives donc la recherche se fait dans une zone restreinte située autour de la position du bloc dans l'image précédente. De plus, nous tenons compte du déplacement du véhicule : par exemple, si l'on sait qu'il se déplace de la gauche vers la droite de l'image (en examinant le vecteur vitesse précédent), on peut placer la zone de recherche toujours autour de la position du bloc dans l'image précédente, mais plutôt à sa droite. De même, selon la vitesse de la voiture, la taille de la zone de recherche est plus ou moins importante.

Précision subpixellique Lorsque les voitures sont trop petites, nous nous sommes rapidement rendus compte que le bloc suivait assez mal la voiture. Le déplacement du bloc n'était pas assez fin. Nous avons donc implémenté une corrélation subpixellique.

Plus précisément, nous avons testé deux méthodes :

- La première consiste à créer des nouveaux blocs par interpolation. Jusqu'à présent, nous avons expliqué que l'on parcourait la fenêtre de recherche en déplaçant le bloc que l'on cherche à faire correspondre d'un nombre entier de pixels verticalement et horizontalement. Si maintenant on souhaite calculer la corrélation qui existe entre le bloc "source" et un bloc "virtuel" situé de façon plus fine (voir Fig. 2.3), il est nécessaire de calculer les intensités de ce second bloc qui n'existe pas réellement. Ceci se fait par interpolation bilinéaire : chaque pixel a un niveau de gris qui est une combinaison linéaire des intensités des 4 pixels qui l'entourent. Cette méthode nécessite donc le choix d'un niveau de précision (1 pixel, 1/2 pixel, 1/4 pixel, etc.).

- Pour la seconde méthode, on commence comme d'habitude par chercher le bloc qui corrèle le mieux avec une précision du pixel. On calcule ensuite l'indice de corrélation obtenu pour les 4 blocs voisins (en haut, en bas, à gauche et à droite). On est alors en possession de cinq valeurs de corrélation. En supposant que la fonction de corrélation est quadratique au voisinage du point traité, on peut alors calculer ces cinq coefficients par interpolation et ainsi trouver la position de son maximum.

Changement d'échelle Les deux blocs n'ont pas forcément la même taille : Dans certaines vidéos, les voitures s'éloignent (resp. se rapprochent) de la caméra et il est nécessaire de diminuer (resp. d'augmenter) la taille du bloc si on veut retrouver les mêmes motifs et textures. Pour ceci, nous cherchons le facteur d'échelle qui permet d'obtenir une "corrélation" maximale. Lorsqu'on souhaite appliquer un facteur d'échelle à un bloc (de 0.9 ou de 1.1 par exemple), il est nécessaire de construire le nouveau bloc à partir du bloc à réduire ou à grossir. Pour cela, nous utilisons une méthode similaire à celle décrite dans le paragraphe ci-dessus : l'intensité de chaque pixel de la nouvelle image est calculée par interpolation bilinéaire à partir des intensités des quatre plus proches voisins dans l'image d'origine.

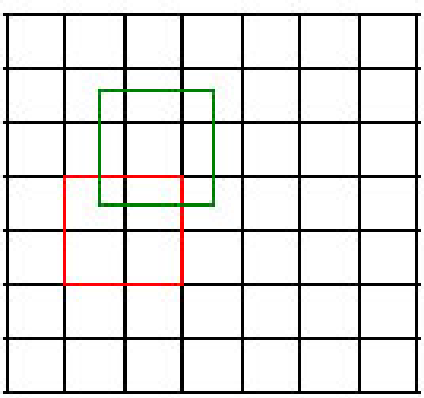

FIG. 2.3 – Précision sub-pixellique : représentation d'un bloc réel (rouge) et d'un bloc virtuel (vert)

2.3 Extraction de coordonnées

2.3.1 Un problème à part entière

Le problème de l'extraction de coordonnées est totalement découplé du problème du suivi pour lequel deux algorithmes distincts ont été proposés. Dans le cadre du problème qui nous était posé, il était certes nécessaire de faire du suivi pour ensuite extraire les coordonnées des voitures, mais plus généralement, on peut se poser le problème de l'extraction de coordonnées quelconques dans une vidéo. La scène étant filmée par une caméra perchée sur un mât à la hauteur H et inclinée de θ par rapport à l'horizontale.

Le principe d'un appareil photo ou d'une caméra étant de capturer une vue tri-dimensionnelle à l'aide d'une vue bi-dimensionnelle (par projection), il y a nécessairement perte d'information. On ne peut donc pas, en général, remonter à la position dans l'espace d'un objet pris en photo, il manquera toujours l'information de profondeur. En effet un ordinateur ne pourra pas savoir que tel objet est au premier plan et que tel autre à l'arrière plan, seul un humain le sait en analysant la photographie.

La solution couramment employée pour contourner ce problème étant d'utiliser deux caméras (principe de vision par stéréoscopie ou stéréovision), nous étions obligé de nous pencher sur une autre solution. Sachant que nous disposions de grandeurs extrinsèques à la caméra (la hauteur du mât où elle était placée, l'inclinaison), nous estimions possible de s'en sortir à l'aide d'une seule caméra.

2.3.2 La modélisation de la caméra

La calibration de caméra

Restait donc à modéliser une caméra. Une technique existe pour cela et nous l'avions vue en cours de "Vision/Traitement d'images". Au lieu d'essayer de récupérer dans de la documentation les paramètres de notre caméra, nous pouvions les estimer grâce à un procédé dit de "calibration" de la caméra. Cela consiste à déterminer au travers d'une matrice de caméra, les paramètres intrinsèques (focale, taille des pixels, asymétrie des pixels, ...) et extrinsèques (position, orientation par rapport à un repère fixe).

En pratique on utilise une mire qu'on place devant la caméra et pour laquelle on connaît toutes les caractéristiques de longueurs, angles, etc. Une fois qu'on dispose de la matrice de caméra, un peu de géométrie projective permet rapidement d'obtenir des résultats.

C'est une technique qui a fait ses preuves et pour laquelle il existe de nombreux logiciels sur Internet qui permettent à chacun de calibrer sa caméra. Cependant l'implémentation d'une telle technique sortait du cadre de ce projet et restait assez contraignante, c'est pourquoi nous avons opté pour une méthode moins précise mais qui avait l'avantage d'être plus simple à mettre en oeuvre et permettait d'arriver à un résultat dans les temps.

Extraction en utilisant de l'optique géométrique

Sans calibrer la caméra il est en effet possible d'arriver à un résultat, l'inconvénient étant qu'il faut rassembler à la main tous les paramètres intrinsèques et extrinsèques. Cela ne nous a donc permis de réaliser le test uniquement sur une vidéo filmée par nous-mêmes.
Cette technique n'est vraisemblablement jamais utilisée car elle introduit de l'imprécision, ce dont nous nous sommes aperçus lorsque nous avons tenté de rassembler les paramètres intrinsèques à la caméra.

Pour comprendre, schématisons grossièrement le fonctionnement d'une caméra :

- une lentille permet de faire converger les rayons lumineux dans le plan focal image de la lentille. Sa caractéristique principale est sa focale c'est à dire la distance entre le centre de la lentille et le plan focal image.

- dans le plan focal image est placé un dispositif qu'on appelle capteur CCD et qu'on retrouve aussi dans tous les appareils qui numérisent comme les appareils photo-numériques et les scanners. Ce capteur permet de convertir l'information lumineuse en courant électrique. Ses caractéristiques sont sa surface et sa définition, c'est-à-dire le nombre de points sur le capteur.

- reste ensuite à effectuer quelques conversions pour générer un fichier compréhensible par un ordinateur (par exemple les points sur le capteur CCD ne sont pas carrés contrairement aux pixels d'un ordinateur).

L'imprécision vient du fait que nous avons été dans l'impossibilité de savoir exactement quelles conversions étaient effectuées à la dernière étape. C'est pour-

quoi les formules que nous présentons ci-dessous sont certainement à corriger (voir figure 2.4).

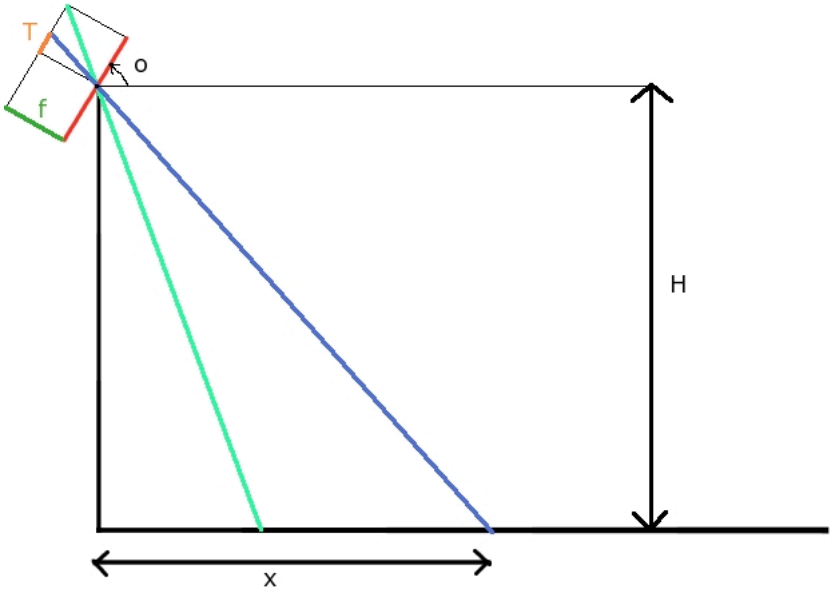

FIG. 2.4 – Un schéma simplifié en vue de profil (route, lentille, plan focal)

Voici ce que donnent les formules pour un point de coordonnées (T_x, T_y) dans l'image finale (origine prise au centre de l'image) :

$$\begin{cases} \dfrac{f\sin\theta - T_y\cos\theta}{y} = \dfrac{f\cos\theta + T_y\sin\theta}{H} \\ \dfrac{T_x}{x} = \dfrac{f}{\sqrt{H^2+y^2}} \end{cases}$$

où on a supposé, faute d'informations suffisantes, que T_x et T_y étaient obtenus simplement en connaissant le *pitch*, c'est à dire la taille réelle d'un pixel sur le capteur CCD tel qu'il est vu sur l'écran :

$$T_i = (\#Px) \times pitch$$

Évidemment, cela peut paraître juste, mais non seulement le *pitch* peut dépendre de la direction (horizontale ou verticale), de plus comme c'est un paramètre qui s'exprime en microns, voire en dixièmes de microns, une erreur de quelques millimètres engendre dans ce cas des résultats totalement faux.

Comme il arrive qu'on ne connaisse pas parfaitement tous les paramètres, on peut notamment lorsqu'un seul est inconnu introduire un peu de calibration dans

la méthode. Par exemple sur la vidéo que nous avons filmée sur une autoroute (plus de détails en 3.1), nous pouvons essayer de calibrer le paramètre inconnu de sorte que la longueur des lignes blanches correspondent à la longueur normalisée.

Cas particulier de l'extraction de vitesses

Dans le cas particulier où on cherche uniquement à estimer la vitesse moyenne des véhicules apparaissant à l'écran, on peut utiliser là encore la connaissance du terrain. Par exemple si on connaît la distance entre deux points sur la route, et si de plus un véhicule passe par ces deux points, on pourra estimer sa vitesse moyenne grâce à la connaissance du nombre d'images qu'il a mis pour effectuer la distance.

Chapitre 3

Tests et résultats

Après avoir implémenté la théorie dans des programmes informatiques, on teste et on donne dans cette dernière partie les résultats des deux algorithmes de *tracking* avant de n'en choisir qu'un pour appliquer l'extraction de coordonnées dont on donne aussi les performances.

3.1 Les vidéos

Pour tester les deux algorithmes décrits précédemment, nous avons utilisé deux "types" de vidéos destinés à des usages très différents :

- une vidéo que nous avons nous-même filmée depuis un pont de l'A4 au niveau d'Emerainville. Nous avons choisi des conditions parfaites (circulation modérée, soleil couché pour diminuer les effets d'ombres, temps sec) et avons filmé une portion plane et rectiligne de l'autoroute. Cette vidéo est considérée comme parfaite et ne reflète pas les difficultés courantes rencontrées sur la plupart des vidéos prises par des caméras de surveillance de la circulation (trafic congestionné, pluie, trajectoire courbe, ...).

- une série de vidéos prises sur le site web de la faculté informatique de Karlsruhe[5] présentant des conditions de circulation plus complexes et donc aussi plus proches de celles qu'utilisera Renault (carrefour, ...). Il est d'ailleurs intéressant de noter que ces vidéos font référence dans le domaine du *tracking*, à la manière de la photographie de Lena pour le traitement d'images.

La première vidéo va nous permettre de tester les capacités de *tracking* sans obstacles et sans bruit, c'est à dire qu'a priori, une fois une voiture détectée, celle-ci ne doit plus être perdue. Des camions et motos de circuler sur l'autoroute donc cette vidéo peut aussi servir à tester le comportement des algorithmes face à des objets de tailles hétérogènes.

Le jeu de vidéos de Karlsruhe permet lui de tester les réactions face aux perturbations par exemple, les obstacles dans le champ de vision, les conditions

de visibilité, les véhicules qui ont une trajectoire non rectiligne ou les chevauchements de deux voitures dus à la perspective. On pourra ainsi vérifier si tel algorithme gère bien les obstructions ou si tel autre sépare bien deux voitures côte à côte.

D'une manière générale, on a toujours vérifié d'abord sur la première vidéo pour voir si une amélioration fonctionnait, et si c'était le cas, on regardait dans quelle mesure cela fonctionnait sur les vidéos de Karlsruhe.
Nous allons donc reprendre chaque algorithme et analyser les problèmes rencontrés et les améliorations ajoutées.

3.2 Algorithme reposant sur le BMA

3.2.1 Vidéo de l'autoroute

Sur la vidéo d'autoroute qui présente des conditions idéales, on ne commence le suivi des véhicules qu'à partir du moment où ceux-ci sont assez éloignés de la caméra. En effet, lorsqu'ils apparaissent dans le bas de l'image, ils sont trop gros par rapport à la taille des blocs. La corrélation n'est donc pas du tout précise et la carte des vecteurs déplacement est très inhomogène, ce qui entraîne une sur-segmentation des véhicules.

Réglage des paramètres Pour cette vidéo, on choisit des blocs de taille maximum égale à 32 pixels et de taille minimum égale à 16 pixels. En effet, même si la taille des véhicules varie, elle est suffisamment importante au début pour justifier une grosse taille de blocs.

Nous avons essayé plusieurs fonctions pour évaluer le critère de ressemblance entre deux blocs sur cette vidéo et le choix évoqué en 2.1.1 a été retenu car il faisait apparaître moins de bruit que la corrélation croisée normalisée et était moins gourmand en temps d'exécution. Comme la vidéo de l'autoroute est de très bonne qualité, cela ne pouvait être que pire pour les autres types de vidéos.

Les résultats On obtient finalement des champs de vecteurs de déplacement assez cohérents sur la moitié haute de l'image, les seuls bémols étant les voitures dont le toit est trop gros ou semblable à la texture de la route et qui ne sont pas détectées comme mobiles. Il faut savoir qu'on rencontre souvent ce problème dans la littérature et qu'il est difficile de s'en débarrasser.

Une fois le lissage du champ de vecteurs effectué, on obtient un résultat convaincant, c'est-à-dire qu'on commence à distinguer les objets sans trop de difficultés. Un détail cependant : plus les voitures s'éloignent, plus leurs vecteurs déplacements sont petits et plus il est difficile de discerner deux objets distincts en regardant uniquement ces vecteurs. On pressent donc qu'il y aura des erreurs lorsque les voitures sont sur le point de disparaître.

Ce sont effectivement ces difficultés que l'on observe : le seuil qui détermine si deux vecteurs déplacement appartiennent à un même objet ne doit pas être le même pour des objets proches et des objets lointains. Il faut en effet bien

comprendre que même si une voiture roule à vitesse constante, son vecteur vitesse varie sur la vidéo à cause de la perspective. C'est seulement l'extraction de coordonnées qui rétablira l'égalité des vecteurs vitesse.

Premier bilan Sur une vidéo considérée comme parfaite, on commence déjà à entrevoir les inconvénients de l'algorithme, à savoir des paramètres qui sont difficiles à régler car ils n'ont pas un sens clair. On s'aperçoit notamment que la taille des blocs représente le plus grand obstacle. En effet, plus celle-ci est petite plus on obtient des résultats mauvais après la première étape du BMA : la carte des vecteurs déplacement est trop bruitée car un petit bloc sans texture peut corréler avec une multitude de voisins. Malheureusement augmenter la taille ne résout pas les problèmes puisqu'on a alors du mal à bien détourer des voitures.

3.2.2 Vidéo de Karlsruhe

Sur la vidéo de Karlsruhe, on peut observer comment se comporte l'algorithme lorsque la vidéo est de moins bonne qualité ou lorsqu'il existe des obstacles dans le champ de vision.
Par exemple les véhicules dont la couleur est trop proche de celle de l'arrière-plan ne sont pas détectés. Des difficultés apparaissent également au niveau des arbres qui occasionnent une sur-segmentation du véhicule en se superposant à lui sur l'image. Lorsque deux voitures sont trop proches et roulent approximativement à la même vitesse - dans le cas d'un dépassement par exemple - elles sont reconnues comme un seul et même véhicule.

Finalement, on peut dégager de cet algorithme de suivi un premier gros désavantage qui est celui du réglage de la taille des blocs. C'est en effet le découpage initial qui conditionne la suite de l'algorithme. Cependant une fois ce désagrément passé, l'algorithme peut assez bien se débrouiller face à des situations difficiles comme des véhicules qui changent de direction ou qui sont légèrement obstrués.

Mais c'est au niveau de l'extraction des coordonnées que l'algorithme pêche réellement : les contours des objets détectés varient énormément d'une image à l'autre, ce qui nous empêche de suivre un point fixe entre deux images consécutives. On pourrait tâcher de suivre le centre de gravité des pixels concernés, mais la surface de l'objet varie elle aussi ainsi que la configuration de ses blocs.

Il semble donc difficile de pouvoir implémenter un algorithme d'extraction à partir du BMA. C'est pourquoi nous avons décidé de concentrer nos efforts sur l'amélioration du second algorithme dont les résultats sont décrits ci-dessous.

3.3 Soustraction de fond & Appariement de blocs

3.3.1 La détection de véhicules

Nous avons vu que cette partie de l'algorithme nécessitait un certain nombre de paramètres à calibrer pour chaque vidéo. Pour chaque voie sur laquelle nous

voulons suivre des véhicules, nous devons définir une zone de détection, des seuils haut et bas de détection, une zone initiale de recherche et les dimensions caractéristiques de la voiture à suivre. Ces paramètres dépendent fortement de la voie considérée et de la vidéo. Pour une meilleure adaptabilité nous avons réalisé une petite interface permettant à l'utilisateur de calibrer le modèle plus facilement.

Une des principales difficultés de la calibration est le choix *a priori* du seuil haut et bas de détection. En effet, comme nous l'avons vu plus haut, la détection des véhicules dans la zone de détection se fait en surveillant la part d'éléments de premier plan présents dans cette zone. Typiquement, les éléments du premier plan correspondent à une voiture.

Enfin, pour obtenir l'arrière-plan, nous avons retenu le calcul de la moyenne des n dernières images, avec n fixé et indépendant des vidéos. Une autre méthode consistait à prendre la moyenne de toutes les images de la vidéo, mais cette méthode a été abandonnée, car le fond obtenu diffère parfois trop du fond courant à l'instant t.

3.3.2 Le suivi des véhicules

Rappelons que pour obtenir les coordonnées de la voiture image par image, nous devons suivre toujours la même partie de la voiture (par exemple la plaque d'immatriculation). La vidéo de Karlsruhe fournit quasiment une vue de dessus, si bien que nous avons décidé de suivre à chaque fois l'intégralité de la voiture. En revanche, pour la vidéo de l'A4, nous ne pouvons pas en faire autant puisque la vue de la voiture change au cours du mouvement du fait de la perspective (en particulier l'avant disparaît lorsque les voitures s'éloignent de la caméra). Dans cette vidéo nous avons donc décidé de ne suivre que l'arrière de la voiture.

Un premier test de l'algorithme a été fait sur la vidéo de l'autoroute. L'algorithme fonctionne correctement et c'est en général toujours l'arrière de la voiture qui est suivi. Le changement d'échelle améliore le suivi mais ralentit l'exécution de l'algorithme. Cependant, lorsque le véhicule devient trop petit, sa texture devient moins présente (problème lié à la vue en perspective) et la voiture est "perdue" par l'algorithme.

L'algorithme a ensuite été testé sur la vidéo de Karlsruhe. Dans un premier temps nous avons testé l'algorithme sans corrélation subpixellique.

Sur l'image sont représentés deux modèles correspondant à deux voies de circulation. Chacun représente la zone de détection (fond bleu) et la zone de recherche initiale (fond transparent). Dans les deux modèles les véhicules se déplacent de droite à gauche. Dans les deux cas, la détection a fonctionné normalement mais le suivi des véhicules est plus difficile. Nous avons pu, avec cette vidéo, mettre le doigt sur certaines difficultés :

- lorsque le véhicule passe derrière un obstacle (par exemple les feux de circulation au centre ou les arbres tout à droite), il arrive que le suivi soit

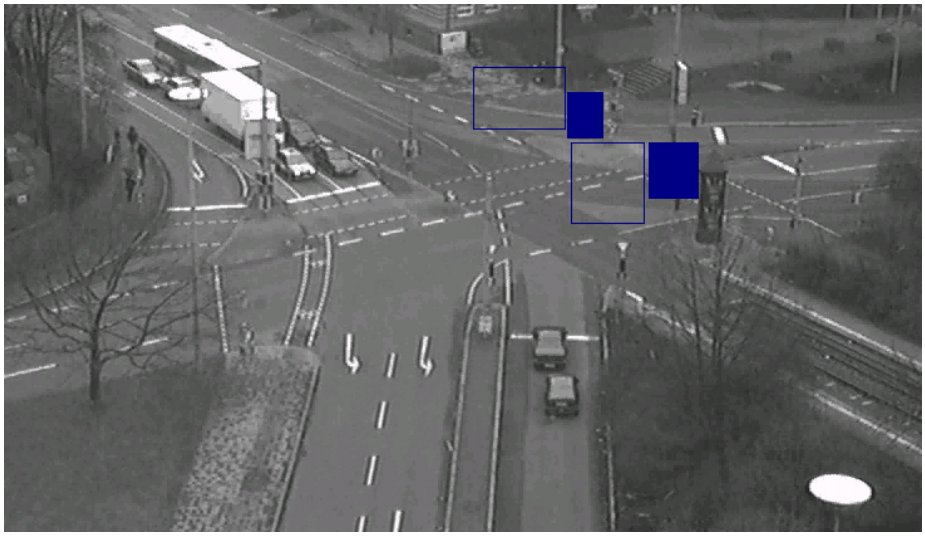

FIG. 3.1 – La calibration de la vidéo de Karlsruhe

bloqué.

– la présence de petits éléments immobiles au sol (lignes blanches) perturbe aussi le déplacement du cadre entourant la voiture. Il est ainsi important que celui-ci contienne le moins possible d'éléments de l'arrière plan.

– dans de nombreux cas encore, le bloc suit de moins en moins bien la voiture pour la perdre finalement totalement.

La corrélation subpixellique, implémentée pour répondre à ces difficultés, a effectivement permis de résoudre en partie ces problèmes. Le suivi fonctionne bien, et l'obstacle des feux est toujours franchi.

3.4 L'extraction en pratique

Comme nous l'avons expliqué en section 2.3, nous avons implémenté l'extraction de coordonnées uniquement sur la vidéo de l'autoroute et pour l'algorithme de soustraction de fond.

On peut observer les résultats à la main en utilisant simplement une interface qui donne la position absolue sur la route d'un point quelconque de l'image, et s'apercevoir que les résultats sont relativement peu précis. En effet nous avons été capable de corriger un peu notre modèle en utilisant les longueurs des lignes discontinues sur l'autoroute, mais on n'arrive pas, par exemple, à obtenir une largeur vraiment constante de la chaussée lorsqu'elle est au premier plan et à l'arrière plan.

Finalement, il s'avère en général indispensable de calibrer la caméra afin d'obtenir une estimation précise des paramètres intrinsèques. Cependant, implémenter une telle méthode ne relevait pas du cadre de notre projet, c'est pourquoi nous avons préféré présenter une méthode certes peu précise mais fonctionnelle.

Bilan

Travail effectué

Nous avons atteint les objectifs initiaux qui nous étaient assignés en ce sens que nous sommes capables de suivre tous les véhicules qui passent devant la caméra : ceux-ci peuvent notamment s'éloigner ou se rapprocher de la caméra et passer derrière certains obstacles. Cependant, il est clair que nous avons implémenté un programme qui est loin de pouvoir être utilisé à grande échelle. Il permet simplement de montrer que l'utilisation future d'un logiciel similaire est possible et nous en avons fait une démonstration à partir de vidéos "choisies". C'est en fait l'objectif qui nous avait été fixé initialement. Il ne s'agissait évidemment pas de programmer en si peu de temps le logiciel qui allait pouvoir être utilisé directement par Renault.

Notre travail trouve ses limites à plusieurs niveaux : concernant l'extraction des coordonnées, nous avons déjà évoqué les difficultés rencontrées dans la section 2.3. De plus le programme n'a pas du tout été réfléchi pour des conditions non optimales (embouteillages, pluie, caméra nocturne, etc.).

Néanmoins, nous avons pensé à quelques améliorations qui pourraient être apporté à notre programme :

- Il pourrait être intéressant que la taille initiale du bloc qui suit un véhicule dépende de ce dernier. Ceci améliorerait le *tracking* grâce à une proportion minimale d'éléments de l'arrière-plan dans le bloc, et permettrait de suivre aussi les camions et les motos.

- Nous avons expliqué que la méthode de soustraction de l'arrière-plan que nous utilisons était assez simple (cf. 2.2.1). Une méthode plus évoluée (modèle de mélanges de gaussiennes [4] par exemple) permettrait peut-être une utilisation de notre programme dans des conditions plus difficiles (phares des voitures allumés, ombres plus importantes, etc.)

- Le suivi de trajectoires courbes présente une autre difficulté : lorsque nous essayons de suivre un véhicule qui tourne, la part du véhicule dans le bloc est plus faible, et l'algorithme a tendance à suivre aussi les éléments extérieurs qui entrent dans le bloc comme les lignes blanches de la chaussée. Ainsi, il faudrait que le bloc puisse aussi tourner.

Point de vue personnel

Nous pouvons à présent dresser un bilan des acquis de cette expérience. Le problème technique à résoudre nous a été présenté dès le début de manière précise, et le déroulement de notre travail a été assez classique : recherche bibliographique (faire un état de l'art, puis choisir l'algorithme le mieux approprié pour résoudre notre problème), implémentation de l'algorithme choisi et analyse des résultats obtenus. Notre recherche a été facilitée par nos entretiens avec M. Paragios, qui est un spécialiste dans le domaine du suivi de véhicules. C'est lui qui nous a d'ailleurs suggéré l'un des deux algorithmes que nous avons implémentés, tout en nous signalant que la résolution de ce problème était très difficile.

Bien que nous ayons simplifié le problème (choix de vidéo de bonne qualité), ainsi que la complexité des algorithmes (notamment la simplification du calcul du modèle d'arrière-plan pour le deuxième algorithme), nous avons, au cours de notre implémentation, été confrontés à de nombreux "petits" problèmes que nous avons toujours su plus ou moins résoudre, mais que nous n'avions pas prévu. L'enseignement que nous en avons tiré est encore une fois qu'il existe un réel décalage entre la théorie (typiquement la description d'un algorithme dans un article et les résultats qui doivent être obtenus) et la pratique (implémentation de cet algorithme et les résultats effectivement obtenus).

Concernant notre organisation, il n'a pas toujours été facile de travailler à quatre. Cet effectif est optimal lorsque l'on peut séparer le travail en deux parties indépendantes (chacune affectée à un binôme) mais cela constitue parfois aussi un handicap. Globalement, ce projet nous a paru à tous les quatre un bon prétexte à expérimenter le travail en groupe (auquel nous devrons nous habituer dans notre vie professionnelle) et nous a permis de prendre un peu de recul vis à vis du contenu purement académique qui est enseigné à l'école.

Bibliographie

[1] Di Stefano, Luigi et Viarani, Enrico. *Vehicle Detection and Tracking Using the Block Matching Algorithm.* University of Bologna, 2000.

[2] Gyaourova, Aglika, Kamath, Chandrika et Cheung, Sen-ching. *Block Matching for Object Tracking.* Lawrence Livermore National Laboratory, 2003.

[3] Piccardi, Massimo. *Background subtraction techniques : a review.* University of Technology Sydney, 2004.

[4] Stauffer, Chris et Grimson, W.E.L. *Adaptive background mixture models for real-time tracking.* Massachusetts Institute of Technology, 1999.

[5] Institut für Algorithmen und Kognitive Systeme. `http://i21www.ira.uka.de/image\_sequences/`. Universität Karlsruhe.