UCLouvain

epl

École polytechnique de Louvain

Conception d'un senseur intégré multimodal pour l'observation des routes

Auteurs: **Kevin De Sousa, Gauthier Rotsart de Hertaing**
Promoteurs: **Benoît Macq, Christophe Craeye**
Lecteurs: **Jean-Benoît Delbrouck, Jean-Didier Legat, Dani Manjah**
Année académique 2020–2021
Master [120] : ingénieur civil électricien

Preface

Ce mémoire est l'aboutissement de cinq années de dur labeur de notre formation d'ingénieur civil électricien. Il nous a permis d'élargir et approfondir nos connaissances, de décortiquer des articles scientifiques de manière critique et d'en synthétiser le contenu. Il nous a appris la persévérance et le travail en équipe à distance. Ce mémoire nous a permis de plonger plus en profondeur dans le domaine de l'intelligence artificielle en découvrant des architectures innovantes, des algorithmes de détection d'objets et plus encore, tout en gardant un aspect électronique avec la simulation du radar. Bref, ce travail de fin d'étude nous a été très bénéfique.

Tout cela n'aurait cependant pas été possible sans l'aide de nos deux promoteurs Benoît Macq et Christophe Craeye qui nous ont soutenus tout au long de cette aventure, que ce soit grâce à leurs conseils lors de moments incertains ou des ressources allouées. Nous remercions aussi Jean-Benoit Delbrouck pour son expertise sur le Transformer qui nous a permis de bien cerner cette architecture, toute l'équipe de *Digital Twin*, François-Xavier Inglese, Jonathan Samelson, Dani Manjah, pour la création de la simulation du carrefour, ainsi que Sébastien Lugan et Kaori Hagihara pour leur aide dans la compréhension des limites des images de synthèse et finalement Benoît Hubert pour son expertise électronique. Nous sommes reconnaissant envers Alexis Duflot pour son aide pour la prise en main du hardware ainsi que pour son TFE.

On remercie nos parents, frères et soeurs ainsi que nos amis pour tout leur soutien apporté au long de l'aventure sans qui cela aurait été difficile en cette période difficile de Covid.

Kevin De Sousa et Gauthier Rotsart de Hertaing, le 13 juin 2021, Louvain-la-Neuve

Abstract

In the field of road observation, multimodal systems are becoming more and more frequent thanks to the duality and complementarity of the information provided by different sensors. The fusion and filtering of data is carried out using a neural network : the Transformer architecture. To train this network, a study site was synthetically reproduced using an UnrealEngine4 game engine in order to obtain images and annotated data for a radar simulation. The extraction of camera and radar data is done respectively by Yolov5 and a search algorithm . The data fusion is performed, tested and compared using three different architectures : the Kalman filter, a Multi-Layer Perceptron (MLP) and the Transformer. Our results show that although the Transformer produces a higher prediction error than the MLP with a 32 times higher execution time, it obtains much smoother and less noisy trajectories, like a Kalman filter, but with a lower error margin. An experimental validation is then performed to validate our results. We notice however that some of our assumptions are too strong, making the results of the transformation less good than expected.

Keywords : camera, radar, Kalman filter, Multi-Layer Perceptron, Transformer, Multimodal sensor,Multiple object tracking

Résumé

Dans le domaine de l'observation des routes, les systèmes multimodaux deviennent de plus en plus fréquents grâce à la dualité et complémentarité des informations fournies par les différents senseurs. La fusion et le filtrage de données sont réalisés grâce à un réseau de neurones : l'architecture Transformer. Pour entraîner ce réseau, un lieu d'étude a été reproduit synthétiquement à l'aide du moteur de jeu UnrealEngine4 afin d'obtenir des images ainsi que des données annotées permettant la réalisation d'une simulation radar. L'extraction des données caméra et radar se fait respectivement par Yolov5 et un algorithme de recherche. La fusion des données est réalisée, testée et comparée à l'aide de trois architectures différentes : le filtre de Kalman, un Perceptron Multi-Couches (MLP) et un Transformer. Nos résultats montrent que bien que le Transformer produise une erreur de prédiction supérieure à celle du MLP avec un temps d'exécution 32 fois supérieur, il obtient des trajectoires beaucoup plus lisses et moins bruitées, telles que celles obtenues avec un filtre de Kalman avec cependant une marge d'erreur inférieure. Une validation expérimentale est ensuite effectuée pour valider nos résultats. On remarque cependant que certaines de nos hypothèses sont trop fortes, rendant les résultats du transformer moins bon qu'espérés.

Mots-clefs : Caméra,Radar, Yolov5 , Filtre de Kalman, MLP, Transformer, Senseur multi-modal, Suivi de trajectoire,

Table des matières

Introduction

En 2019, il y a eu 19601 accidents corporels sur des routes limitées à 50 [km/h] et 6941 sur des routes limitées à 70 [km/h] d'après [1]. En outre, selon cette même source, il y aurait eu 12035 accidents sur des intersections. Pour comprendre les causes de ces accidents ainsi qu'analyser le comportement des conducteurs sur la route, des systèmes de surveillance routière sont mis en place, dotés d'un ou de plusieurs senseurs. En plus, depuis quelques années, l'intelligence artificielle s'est largement répandue et démocratisée dans beaucoup de domaines, permettant d'obtenir des résultats difficilement atteignables par d'autres procédés. C'est dans ce cadre que ce travail va se focaliser sur l'analyse de la trajectoire de cibles à l'aide d'un système bimodal radar-caméra grâce à l'implémentation d'un réseau de neurones particulier. Celui-ci devant être capable de fusionner des données provenant de deux senseurs, entraîner correctement un tel réseau nécessite une grande quantité de données.

Afin d'acquérir suffisamment de données mais aussi de se confronter à un cas pratique, un lieu d'étude a été reproduit synthétiquement par l'expert François-Xavier Inglese à l'aide du moteur de jeu UnrealEngine4 permettant de simuler du trafic routier. Cette simulation fournit des images synthétiques ainsi qu'un fichier avec les données précises telles que la position des véhicules dans l'espace et leur vitesse. Grâce à ce fichier, la simulation d'un radar Doppler a été effectuée, permettant de simuler entièrement un système bimodal radar-caméra observant un carrefour.

Une fois les trajectoires des différents véhicules extraites des données simulées, des architectures ont été implémentées afin de filtrer et d'affiner les résultats. Pour ce faire, un filtre de Kalman a été utilisé, permettant à la fois une fusion linéaire des données mais servant également de repère pour la comparaison des résultats avec des architectures neuronales de type Perceptron Multi-Couches (MLP) et Transformer. Ce dernier est une nouvelle forme d'intelligence artificielle utilisant un mécanisme d'attention pour fusionner les informations extraites, contrairement à bon nombre d'architectures neuronales utilisant des couches convolutionnelles. Dans le cadre

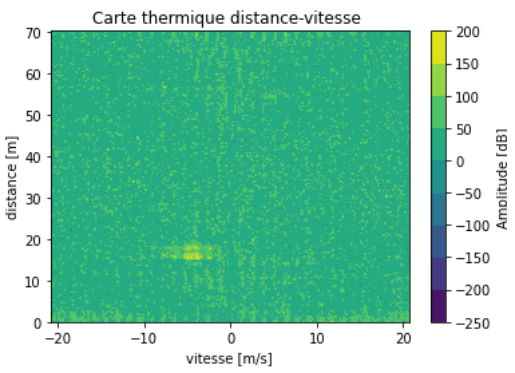
de ce travail, le MLP implémenté sert de référence pour les résultats produits par un réseau de neurones. Ainsi, nous pourrons comparer les performances entre une architecture linéaire (filtre de Kalman) et non linéaire (MLP, Transformer), mais également entre deux réseaux neuronaux. Etant donné qu'une simulation comporte toujours des hypothèses, celle-ci ne sait jamais refléter parfaitement la réalité. Dès lors, une validation expérimentale a été réalisée pour évaluer la pertinence des algorithmes et de leurs résultats.

Dans ce travail, les Chapitres 1 et 2 traiteront respectivement de l'élaboration des simulations radar et caméra, qui seront basées sur des exemples réels. Ensuite, le Chapitre 3 sera consacré au développement d'algorithmes d'extraction des données simulées et le Chapitre 4 exposera les différentes architectures utilisées pour fusionner et filtrer les données extraites. Les avantages et inconvénients de chaque méthode seront présentés. Finalement, le Chapitre 5 vérifiera le fonctionnement des algorithmes proposés dans des cas réels et le Chapitre 6 exposera des pistes d'améliorations possibles.

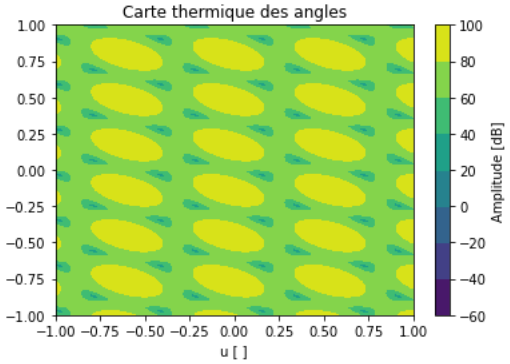
Chapitre 1

Simulation radar

Un radar est un dispositif permettant de localiser une masse métallique en mouvement. Plus précisément, nous allons utiliser et modéliser un radar Doppler qui, comme son nom l'indique, utilise l'effet Doppler pour mesurer la vitesse radiale d'une cible. Notons qu'une vitesse positive est définie quand un véhicule s'éloigne du système radar-caméra. Concrètement, une acquisition de données radars nous donne les deux heatmaps (carte thermique en français) suivantes :



(a) Cette image est une heatmap permettant de représenter la distance et la vitesse de l'ensemble des cibles détectées à un instant t. Un troisième axe permet de représenter l'intensité du signal reçu.



(b) Cette image est une heatmap permettant de représenter la position spatiale d'une cible. Couramment, ceci est représenté dans la base des coordonnées (u, v) qui sera présentée à la Section 1.3. Un troisième axe permet de représenter l'intensité du signal.

FIGURE 1.1 – Exemple d'acquisition de données radars

Sur la Figure 1.1a, on constate un amas de points avec une amplitude supérieure à 200 [dB] se situant à une distance de 18 [m] du radar et s'en rapprochant à une vitesse de -6 [m/s]. Il s'agit d'un véhicule capté par le radar. Sur cet amas, on observe deux bandes lumineuses horizontales. Il s'agit d'un phénomène nommé spectre micro-Doppler et dû au mouvement harmonique des roues du véhicule. Cet étalement horizontal peut être plus ou moins prononcé en fonction de l'orientation de l'axe de la roue par rapport au radar, comme expliqué à la Section 1.1.3. L'étalement vertical du spectre représente quant à lui la distance inter-essieu du véhicule. En outre, sur cette Figure, on peut observer des points ponctuels d'amplitude élevée correspondant à du bruit. Par ailleurs, sur la Figure 1.1b, on y constate des maxima périodiques. Cette périodicité correspond aux ambiguïtés créées par l'espacement entre les antennes du radar, ce qui sera expliqué à la Section 1.1.5.

Dans les sections qui suivent, nous allons modéliser le fonctionnement physique d'un radar en tenant compte des phénomènes observés à la Figure 1.1. Ensuite, nous décrirons comment simuler et obtenir la heatmap distance-vitesse ainsi que celle relative aux angles.

1.1 Fonctionnement physique d'un radar

Les résultats et le raisonnement s'inspirent de ce qui a été fait dans [2] et [3].

1.1.1 Le principe

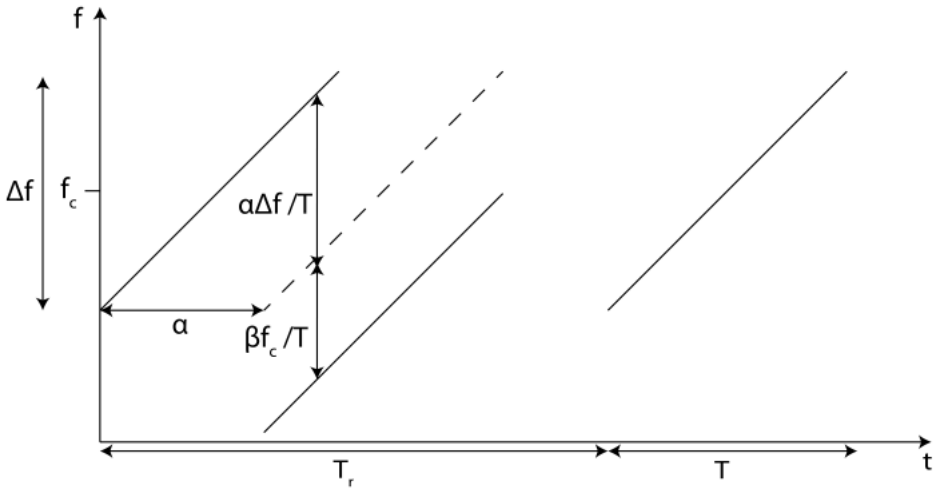


FIGURE 1.2 – Ce schéma, provenant de [2], représente deux rampes de fréquences dans le plan temps-fréquence. Le paramètre α représente le temps de trajet de l'onde envoyée par le radar tandis que le paramètre β représente le décalage de fréquences dû à l'effet Doppler. La droite en pointillé correspond au signal utile reçu s'il n'y avait pas ce décalage de fréquences.

Le radar simulé est un radar du type FMCW (Frequency-Modulated Continuous Waves), ondes continues à fréquence modulée en français. Celui-ci envoie N_2 rampes de fréquences, chacune étant une onde dont la fréquence varie linéairement en fonction du temps comme illustré à la Figure 1.2. A la réception des signaux, N_1 échantillons sont pris par rampe ce qui implique que, pour chaque impulsion de signal envoyée, le radar fournit $N_1 \times N_2$ échantillons. Par la suite, on considère une onde de fréquence centrale f_c dont la fréquence varie linéairement sur une bande passante Δf.

En pratique, la distance d'un objet à un radar peut être retrouvée grâce au temps de trajet de l'onde entre sa transmission et sa réception. Cela se traduit par :

$$d = \frac{2T_r}{c} \tag{1.1}$$

où c est la vitesse de la lumière et T_r le temps de trajet aller-retour de l'onde.

De même, la vitesse d'un objet peut être retrouvée grâce au décalage fréquentiel de l'onde par rapport à la fréquence initiale. Cet effet est connu sous le nom d'effet Doppler. Mathématiquement, cela peut s'écrire comme :

$$\Delta F = f_r - f_c = \left(2v_{cible}\frac{c}{c - v_{cible}}\right) f_c \tag{1.2}$$

avec v_{cible}, la vitesse de la cible, positive si elle se rapproche du radar, et f_r la fréquence reçue.

1.1.2 Modélisation mathématique du radar

Une rampe de fréquences, appelée chirp et notée $s_0(t)$, est modélisée mathématiquement par la forme suivante :

$$s_0(t) = A\cos\left(\left(\omega_0 - \frac{\Delta\omega}{2} + \frac{\Delta\omega t}{2}\right)t\right) rect_T(t) \tag{1.3}$$

Le signal émis par l'antenne émettrice étant composé de P chirps émis à intervalle régulier T_r, il s'écrit donc comme :

$$s(t) = \sum_{p=0}^{P-1} s_0(t - pT_r) \tag{1.4}$$

Concrètement, le signal émis $s(t)$ est réfléchi en entrant en contact avec une cible. Cet écho est alors capté par les antennes réceptrices du radar et renferme les informations nécessaires à la détermination de la distance et de la vitesse de la cible. Ce sont les paramètres α et β, définis à la Figure 1.2, et que nous regrouperons à travers un unique paramètre θ pour ne pas alourdir les notations. Le signal est ensuite démodulé à l'aide d'un mélangeur et d'un filtre passe-bas et les composantes de Rice du signal en sont extraites. Celles-ci sont alors échantillonnées à une fréquence de deux fois la fréquence maximale du signal afin de respecter le théorème de Nyquist-Shannon. Le signal reçu et filtré $r_f[n]$ est toutefois corrompu par du bruit modélisé comme étant du Bruit Blanc Gaussien Additif (BBGA). On a donc :

$$r_f[n] = s[n, \theta] + N_f[n] \tag{1.5}$$

où $s[n, \theta]$ représente la partie utile du signal et $N_f[n]$ le bruit qui le corrompt.

Comme expliqué dans [2] et [3], le signal $r_f[n]$ est aléatoire et peut être décrit par sa fonction de densité de probabilité. On y montre que l'information concernant la distance objet-radar et celle concernant la vitesse de la cible peuvent être obtenues en maximisant une corrélation entre le signal reçu et le signal filtré. Comme dit précédemment, le paramètre θ regroupe en fait deux paramètres : α contenant l'information de la distance de la cible par rapport au radar et β contenant l'information liée à la vitesse de la cible. Cette fonction de corrélation s'écrit comme :

$$C(\hat{\alpha}, \hat{\beta}) = \left| \sum_{n_2=0}^{N_{2,max}-1} e^{j\frac{\Delta\omega}{T}\hat{\alpha}n_2T_s} \sum_{n_1=0}^{N_{1,max}-1} e^{jn_1T_s\omega_0\hat{\beta}} e_r(n_1N_r + n_2) \right|^2 \tag{1.6}$$

En définissant $\Omega_{\hat{\alpha}} = \frac{\Delta\omega}{T}\hat{\alpha}T_s$ et $\Omega_{\hat{\beta}} = N_rT_s\omega_0\hat{\beta}$, on obtient la corrélation suivante à maximiser :

$$C(\hat{\alpha}, \hat{\beta}) = \left| \sum_{n_2=0}^{N_{2,max}-1} e^{j\Omega_{\hat{\alpha}}n_2} \sum_{n_1=0}^{N_{1,max}-1} e^{jn_1\Omega_{\hat{\beta}}} e_r(n_1N_r + n_2) \right|^2 \tag{1.7}$$

Lorsque cette corrélation est maximale, cela correspond à une cible détectée. Une manière efficace d'optimiser cette corrélation, et donc d'en déduire la distance radar-objet ainsi que la vitesse de la cible, est d'appliquer une transformée de Fourier à deux dimensions afin d'en obtenir les maxima locaux. C'est sur ce principe que nous allons construire la simulation du radar. En effet, pour ne pas sortir du cadre de ce travail, nous nous sommes restreints à simuler cette corrélation sans tenir compte des signaux physiques théoriquement reçus, étant donné qu'il est extrêmement difficile de prendre en compte l'ensemble des effets possibles tels que les aspérités des véhicules lors de la réflexion, les multiples réflexions, etc.

1.1.3 Modélisation du spectre micro-Doppler

L'étalement du spectre sur l'axe horizontal de la Figure 1.1a est dû à un phénomène appelé spectre micro-Doppler. Cet effet est créé car la vitesse perçue par le radar n'est pas la même sur l'ensemble de la cible. Cette différence de fréquence se traduit dans le domaine spectral par un étalement de la fréquence autour de la fréquence Doppler correspondant à la vitesse.

Prenons par exemple la roue d'un véhicule qui roule à une vitesse v en face de l'axe de vue d'un radar. Si on prend en compte la condition de roulement sans glissement, on sait que la vitesse au point de contact du sol sera nulle tandis que la vitesse à son extrémité supérieure sera le double de la vitesse v. Cette différence de vitesse correspond dans le domaine spectral à un étalement de la fréquence autour

de la fréquence centrale donnée par l'effet Doppler pour la vitesse v comme illustré à la Figure 1.3.

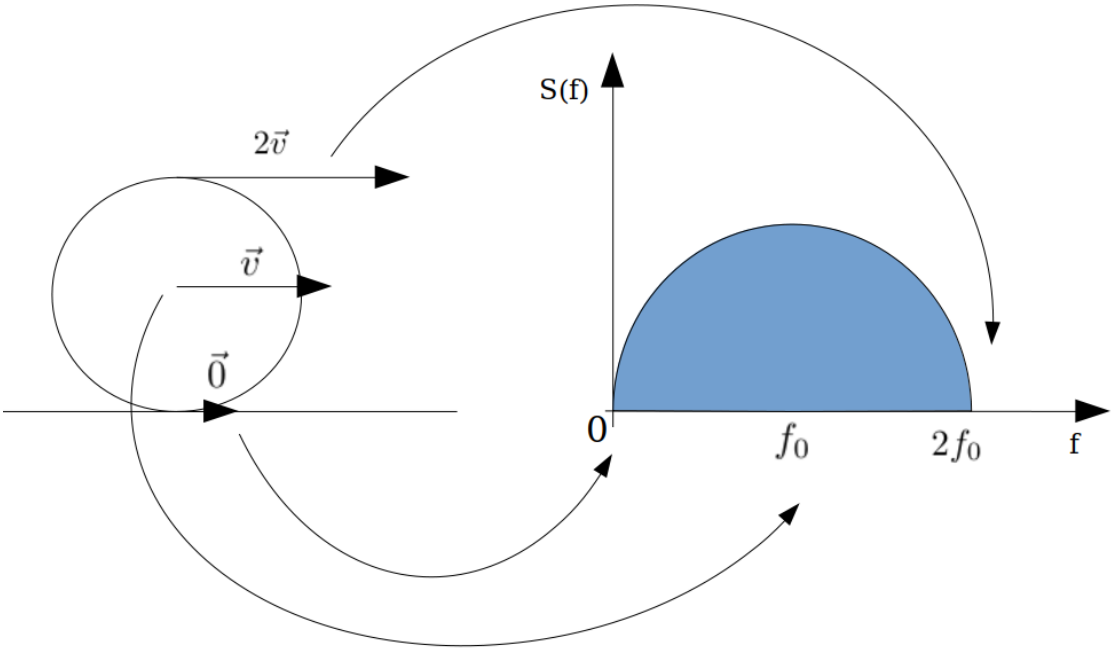


FIGURE 1.3 – Etalement du spectre micro-Doppler [4]

Dans cette section, on montrera l'expression analytique du spectre micro-Doppler d'une roue dans la cas où le radar n'est pas aligné avec l'axe de la roue. En se basant sur les notations de la Figure 1.4, on émet plusieurs hypothèses permettant de simplifier les développements futurs :

— on se trouve en champ lointain ;

— le radar est monochromatique, donc à longueur d'onde λ constante.

Du fait que le radar est monochromatique, on peut montrer que le signal reçu s'écrit comme :

$$s(\theta, \phi) = \frac{A}{R^2(\theta, \varphi)} e^{-2jkR(\theta,\varphi)} \tag{1.8}$$

où A est une constante.

En calculant la distance $R(\theta, \varphi)$, il est possible de calculer le spectre de ce signal. Le développement est réalisé dans [2]. L'expression finale est reprise ci-dessous :

$$S(\omega) = \sum_{n=-\infty}^{\infty} \frac{A' j^n J_n(2ka)}{2\pi} \delta(\omega_0 - n\Omega) \tag{1.9}$$

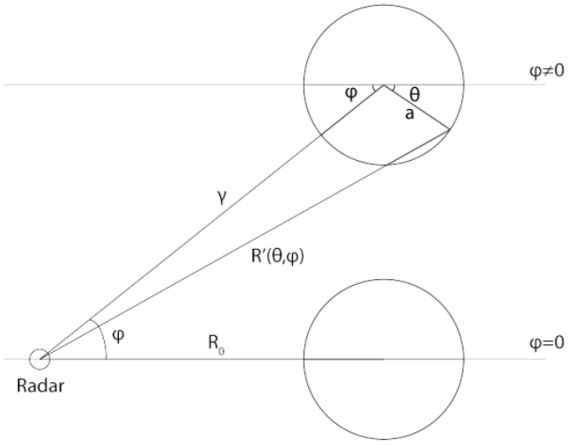


FIGURE 1.4 – Modélisation de l'effet micro-Doppler pour une roue de rayon a. La distance du radar au centre de la roue est notée R_0 et la distance du radar au point étudié de la roue est notée R. [2]

où Ω est la vitesse angulaire de la roue et $A' = \frac{A\cos^2(\varphi)}{R_0^2} e^{2jk\frac{R_0}{\cos(\varphi)}}$

De l'expression analytique du spectre, deux observations sont importantes pour la simulation :

1. L'Equation (1.9) correspond à un ensemble de raies à pulsation régulière $\omega_0 - n\Omega$.
2. Lorsque $\varphi = \frac{\pi}{2}$, A' = 0 puisque $cos(\varphi) = 0$. Dés lors, on ne pourra pas observer de spectre micro-Doppler quand le véhicule se déplace dans l'axe perpendiculaire au radar.

1.1.4 Direction du signal

Cette sous-section va s'intéresser aux effets qui se produisent lors de la réception d'un signal venant d'une direction $\hat{u}$ quelconque. Pour avoir une idée physique de ce qui se passe, commençons par prendre un cas en deux dimensions où le signal provient d'une direction θ. Nous considérons un réseau d'antennes espacées linéairement comme à la Figure 1.5 :

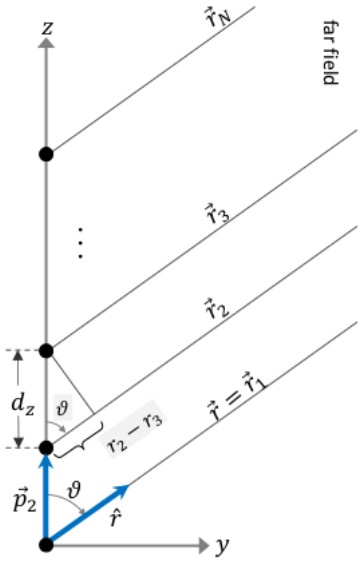


FIGURE 1.5 – Réseau d'antennes dans le plan OYZ [5]

Nous supposons l'hypothèse du champ lointain telle que $|\vec{r}| > \frac{L^2}{\lambda}$ où L est la longueur caractéristique d'une antenne. Cette hypothèse nous permet de supposer que l'onde électromagnétique $\vec{E}$ reçue peut être considérée comme plane. Par conséquent, les signaux reçus par les différentes antennes réceptrices ont une direction de déplacement parallèle entre eux. Cependant, les signaux captés à un instant t entre deux antennes ne seront pas identiques étant donné qu'ils ont chacun une distance différente à parcourir. A la Figure 1.5, cela est représenté par le décalage $\Delta r = r_2 - r_3$. Cette différence de distance à parcourir se traduit donc par un délai à la réception, que l'on peut aussi écrire comme un déphasage $\Delta\Phi$. En effet, le délai entre deux signaux Δt s'exprime comme :

$$\Delta t = \frac{\Delta r}{c}\Delta\Phi \tag{1.10}$$

$$= \frac{2\pi\Delta r}{\lambda} \tag{1.11}$$

Inversément, la différence de distance parcourue Δr est donnée par :

$$\Delta r = c\Delta t \tag{1.12}$$

$$= \frac{\lambda\Delta\Phi}{2\pi} \tag{1.13}$$

Par ailleurs, le déphasage entre 2 antennes est décrit par

$$\Delta\Phi = -kd_z(i-1)\cos(\theta) \tag{1.14}$$

C'est grâce à cette information qu'il est possible de déterminer la provenance du signal. A la réception, chaque antenne i va réceptionner les différents signaux mais avec un déphasage différent. On a ainsi pour chaque antenne :

$$E_i(\theta) = Ee^{j\Delta\Phi_i} \tag{1.15}$$

Ce cas 2D est généralisable à un cas 3D, c'est-à-dire en rajoutant un réseau d'antennes sur un nouvel axe. Pour déterminer la provenance du signal, il suffit de compenser ce déphasage pour chaque signal reçu et de trouver le maximum de tous les signaux reçus. Pour cela, le moyen le plus simple et exhaustif consiste à compenser le déphasage $\Delta\Phi$ dans toutes les directions possibles. Un inconvénient de cette méthode est l'apparition d'ambiguïtés, correspondant à la présence de multiples maxima locaux, comme observés à la Figure 1.1b.

$$S(t) = \sum_{i=1}^{N} E_i e^{-j\Phi_i} \tag{1.16}$$

où $\Phi_i = \frac{2\pi}{\lambda} d_i u_i$ réprésente toutes les directions $\hat{u}$ possibles.

L'échantillon du champ électrique E_i est choisi à l'aide de la carte thermique distance-vitesse. En effet, si un véhicule se trouve au i$^{\text{ème}}$ échantillonage de la j$^{\text{ème}}$ rampe, on reprendra la valeur du champ électrique E_i de cet échantillon pour chaque antenne.

1.1.5 Apparition des ambiguïtés

Sur la Figure 1.1b, on peut observer que les ambiguïtés sont périodiques. Par l'Equation 1.16, on comprend qu'elles viennent de l'argument de l'exponentielle qui est périodique de période 2π. Montrons-le mathématiquement pour le cas 2D :

$$kd_z \cos(\theta) + 2\pi s = kd_z \cos(\theta_0) \tag{1.17}$$

$$\cos(\theta) + \frac{s\lambda}{d_z} = \cos(\theta_0) \tag{1.18}$$

En posant $\Delta = \cos(\theta_0) - \cos(\theta)$, on a :

$$\Delta = s\frac{\lambda}{d_z} \qquad \text{avec } s = ..., -1, 0, 1, ... \tag{1.19}$$

Ainsi, l'argument de l'exponentielle présente dans l'Equation 1.16 est périodique. Par ailleurs, sur une heatmap, les ambiguïtés seront en général exprimées dans la base des cosinus directeurs. Etant donné que la largeur de cet intervalle vaut 2 et

que chaque ambiguïté est distante de $\frac{\lambda}{d}$ (avec $d = d_z$ ou $d = d_x$ suivant la direction choisie), le nombre d'ambiguïtés est donné par :

$$N = \frac{2d}{\lambda} \tag{1.20}$$

où N est arrondi à l'entier le plus bas.

Ces ambiguïtés constituent une caractéristique fondamentale d'un radar Doppler. Le chapitre 3 sera dédié à la levée de ces ambiguïtés. Finalement, notons que pour ne pas relever d'ambiguïté, il faudrait que la distance entre les antennes soient inférieure à la longueur d'onde ou, inversement, que la longueur d'onde soit plus grande que la distance entre les antennes.

1.2 Heatmap distance-vitesse

Sur base de la Figure 1.1, nous constatons que la simulation d'un radar Doppler (tel que celui utilisé) observant N cibles doit fournir $N + 1$ heatmaps : une image représentant l'ensemble des cibles dans le plan distance-vitesse ainsi qu'une image par cible représentant les ambiguïtés relatives à celle-ci (soit N images en tout). Dans le cas de la simulation de la caméra, nous avons accès à chaque instant aux coordonnées (X, Y, Z) du centre de masse des cibles. En les ramenant dans le repère du radar, nous savons que ce point sera situé à une distance d donnée par :

$$d = \sqrt{(X - X_c)^2 + (Y - Y_c)^2 + (Z - Z_c)^2} \tag{1.21}$$

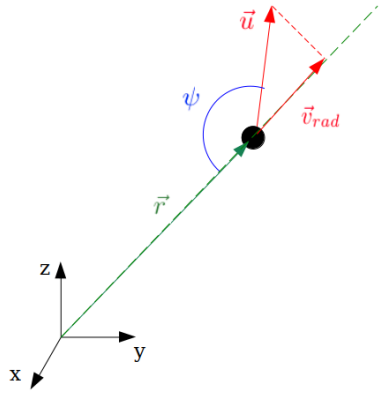


FIGURE 1.6 – Projection du vecteur vitesse $\vec{v}$ sur le vecteur de déplacement $\vec{r}$

La vitesse étant mesurée grâce à l'effet Doppler, seule la composante radiale de celle-ci est mesurée. Ainsi, afin de trouver cette vitesse radiale, il est nécessaire de faire une projection orthogonale du vecteur $\vec{v}$ sur le vecteur $\vec{r}$ comme montré à la Figure 1.6 de telle manière que :

$$v_{rad} = \|v\| \cos(\psi) = \frac{\vec{r} \cdot \vec{u}}{\|r\|} \text{ où } \|v\| = v_{abs} \tag{1.22}$$

où $\vec{u}$ correspond au vecteur unitaire de direction de déplacement de la cible.

Désormais, nous avons tous les outils en main pour modéliser des données radars, en particulier le spectre micro-Doppler qui est caractéristique de la heatmap distance-vitesse.

1.2.1 Modélisation des véhicules et détermination du point spéculaire

Les données de la simulation caméra nous donnent des informations importantes sur le véhicule telles que sa positon, son orientation ou encore ses dimensions. Il nous est dès lors possible de simuler la position du châssis par rapport à la position du radar. Les hypothèses suivantes sont faites :

- La situation est représentée sur le plan OXY ;
- Le châssis est représenté par un polygone de 8 cotés dont 2 cotés opposés non adjacents sont de longueur a et de largeur b avec une rotation γ. Les dimensions a, b sont déterminées par le modèle du véhicule tel qu'illustré à la Figure 1.7.

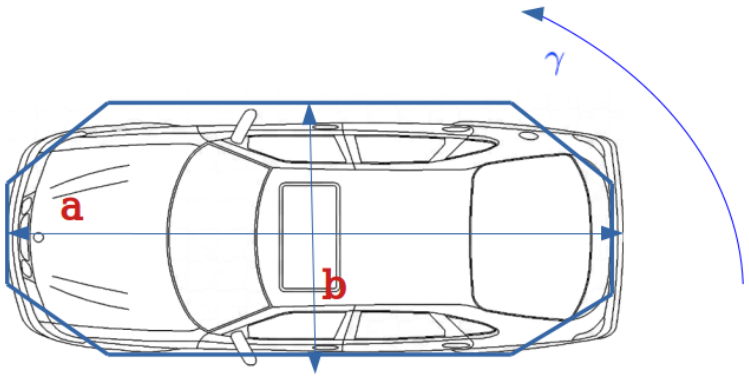
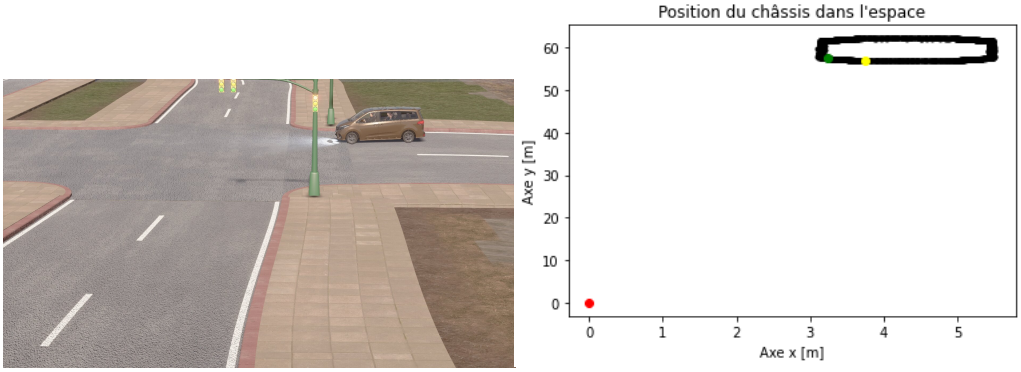


FIGURE 1.7 – Véhicule modélisé par un polygone à 8 faces

Concrètement, les ondes envoyées par le radar vont entrer en contact avec une masse métallique, être réfléchies et ensuite être captées par les antennes réceptrices. Pour ne pas tenir compte de toutes les réflexions multiples et de l'ensemble des phénomènes de diffraction, seul le point spéculaire va être pris en compte. Celui-ci est défini comme étant le point perpendiculaire à la surface du châssis réfléchissant l'onde avec la plus grande intensité vers le radar. Ceci est illustré à la Figure suivante pour un cas concret :



(a) Exemple de scène observée par la caméra.

(b) Représentation de la scène pour le radar (en rouge) : le véhicule est un octogone et le point spéculaire est représenté en jaune.

FIGURE 1.8 – Modélisation d'un véhicule vu par le radar

En connaissant la position du centre de gravité du véhicule dans l'espace ainsi que ses dimensions, le véhicule peut être représenté dans le plan OXY avec comme origine le système radar-caméra tel que représenté sur la Figure ci-dessus. En calculant toutes les normales à la surface du châssis, celle passant le plus proche de l'origine du repère est sélectionnée comme étant le point spéculaire. Ce nouveau point correspondra dorénavant à la positon du véhicule modélisé sur la heatmap. Autrement dit, la distance de ce point au radar ainsi que sa vitesse correspondront à la distance objet-radar et à la vitesse de la cible.

1.2.2 Création du nuage de points par véhicule

Après avoir déterminé le point spéculaire de chaque véhicule, on peut générer la heatmap distance-vitesse. Pour ce faire, on modélise séparément chaque véhicule par un nuage de points dont la distribution suit une loi normale multivariée de moyenne $\mu = \begin{bmatrix} d & v_{rad} \end{bmatrix}$ et de covariance $\Sigma = \begin{bmatrix} d_{res} & 0 \\ 0 & v_{res} \end{bmatrix}$. C'est-à-dire :

$$\mathcal{N}(\mu, \Sigma) \sim \frac{1}{(2\pi)\det(\Sigma)^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)\Sigma^{-1}(x-\mu)\right) \tag{1.23}$$

où res_d et res_v correspondent respectivement à la résolution en distance et en vitesse du radar, d représente la distance du point spéculaire au radar et v_{rad} la vitesse radiale de la cible. Par ailleurs, x est une matrice de taille 256×256 modélisant le signal reçu et échantillonné après optimisation de la corrélation.

Chaque véhicule est représenté sur la carte thermique par un amas de points. Comme expliqué à la Section 1.1.3, la vitesse du véhicule n'est pas le seul élément perçu par le radar. En plus du châssis du véhicule dont le corps se déplace à la même vitesse v, on peut considérer les roues dont chaque point se déplace à une vitesse différente. Dans l'Annexe A, l'étude des expressions des vitesses maximale et minimale de la cinématique des roues a été faite . On retrouve les deux expressions suivantes :

$$v_{min} = -v_{abs}\left((1 - \frac{u}{\sqrt{1+u^2}})\sin\theta\cos(\phi-\gamma) + \frac{1}{\sqrt{1+u^2}}\cos\theta\right) \tag{1.24}$$

$$v_{max} = -v_{abs}\left((1 + \frac{u}{\sqrt{1+u^2}})\sin\theta\cos(\phi-\gamma) - \frac{1}{\sqrt{1+u^2}}\cos\theta\right) \tag{1.25}$$

avec $u = \tan\theta\cos(\phi-\gamma)$.

Ces deux vitesses représentent respectivement les vitesses minimale et maximale perçues par le radar, indépendamment du rayon de la roue, permettant de déterminer l'étalement du spectre sur la carte thermique. En conséquence, nous avons la

largeur de l'étalement de vitesse correspondant théoriquement au micro-Doppler. On note que cette bande n'est pas toujours centrée en v_{rad}.

Pour simuler l'étalement du spectre sur la distance correspondante aux zones où il y a réflexion de l'onde sur la carlingue du véhicule, on convolue la carte thermique avec une noyau K de taille $N \times 5$ où N est la taille effective du véhicule divisée par la résolution en distance. Celle-ci est donnée par l'équation suivante :

$$N = \frac{1}{c} \frac{a\cos(\gamma) + b\sin(\gamma)}{d_{res}} \tag{1.26}$$

où c est un facteur de pondération permettant de limiter la taille effective du véhicule.

Un noyau K de taille 5x5 avec $X = \frac{1}{5N} = \frac{1}{25}$ aura l'allure suivante :

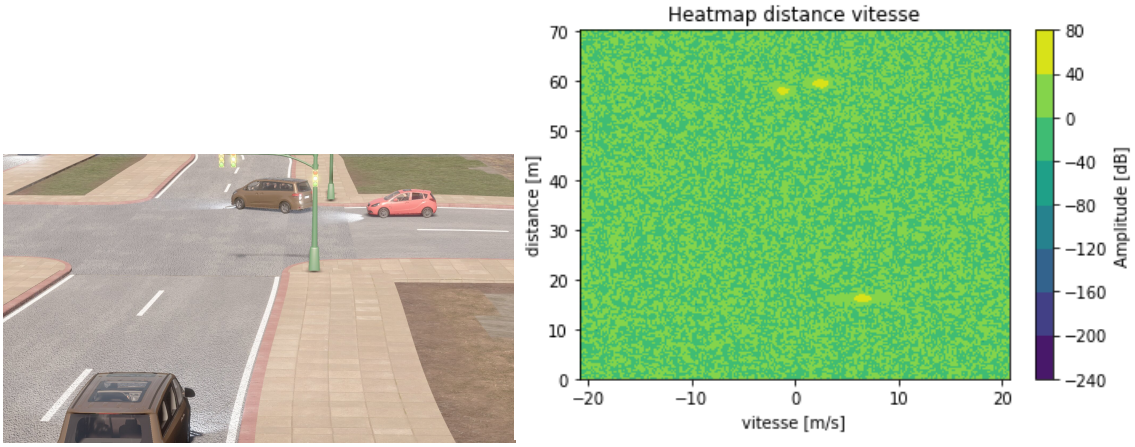
$$K = \begin{bmatrix} Xe^{-4} & Xe^{-2} & X & Xe^{-2} & Xe^{-4} \\ 0 & 0 & X & 0 & 0 \\ 0 & 0 & X & 0 & 0 \\ 0 & 0 & X & 0 & 0 \\ Xe^{-4} & Xe^{-1} & X & Xe^{-2} & Xe^{-4} \end{bmatrix} \tag{1.27}$$

Pour modéliser les amplitudes, il a été décidé de prendre une amplitude arbitraire étant donné que les amplitudes présentes lors des mesures réelles varient trop en fonction d'une multitude de variables. Pour combler cette information, il a alors été décidé de normaliser les données avant leur traitement afin de contourner ce problème. La normalisation est effectuée de la manière suivante :

$$X_{norm} = \frac{X - \mu}{\sigma} \tag{1.28}$$

avec μ la moyenne des amplitudes et σ l'écart-type de l'amplitude.

Sur base de cela, nous obtenons par exemple la carte thermique suivante :



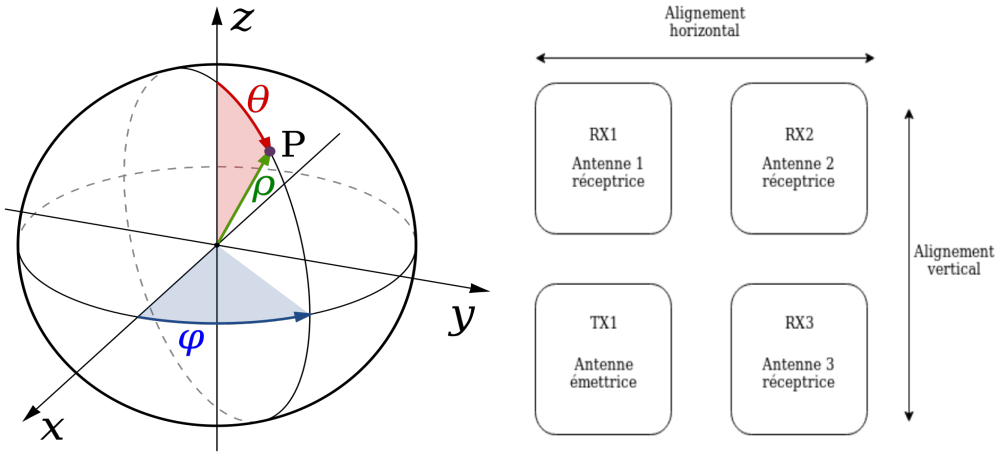
(a) Vue observée par la caméra.

(b) Sur cette heatmap, on peut observer la présence de trois véhicules, chacun modélisé par un amas de points. On peut également observer le spectre micro-Doppler correspondant à l'étalement en vitesse. L'intensité est quant à elle normalisée.

FIGURE 1.9 – Exemple de heatmap distance-vitesse simulée

1.3 Heatmap des angles

Dans cette section, on va expliquer comment générer la heatmap des angles. Tout d'abord, précisons qu'une carte thermique pour la détermination de la position d'un véhicule n'est disponible que si l'amas de points du véhicule dans la carte thermique de la distance/vitesse est bien distinct/éloigné des autres véhicules. Dans le cas contraire, la position angulaire du véhicule sera mal déterminée.



(a) Repère sphérique utilisé [6]

(b) Alignement physique des antennes dans le plan OYZ

FIGURE 1.10 – Repère utilisé et alignement des antennes. [7]

Sur la Figure 1.10a, le repère utilisé pour se repérer dans l'espace est décrit. Le réseau d'antennes est placé dans le plan OYZ avec l'antenne 2 comme point d'origine. Etant donné que le champ de vue du radar est limité à une demi-sphère dans les $x \geq 0$, il a été décidé de passer en coordonnées (u, v), définies à l'aide des opérations suivantes :

$$u = \sin(\varphi)\sin(\theta) \tag{1.29}$$

$$v = \cos(\theta) \tag{1.30}$$

Cette transformation permet de réduire la complexité des calculs en enlevant notamment la prise en compte des quadrants. Le zone décrite par la heatmap sera par ailleurs bornée à $[-1, 1]$ dans les deux dimensions.

Comme expliqué à la sous-section 1.1.4, le signal reçu par un réseau d'antennes composé de N antennes d'une direction particulière (θ, φ) subit un déphasage. Celui-ci varie en fonction de la direction de sa provenance. Afin de la déterminer, on cherche à compenser ce déphasage pour l'amplitude du champ électrique $E(t)$ reçue. Ce déphasage, nommé Φ dans l'Equation (1.16), est exprimé comme $\Phi = k(d_{y,i}u + d_{z,i}v)$ où u et v correspondent à toutes les directions possibles et $d_{y,i}$ et $d_{z,i}$ aux distances des antennes par rapport à l'antenne de référence. L'expression mathématique de la corrélation à maximiser s'écrit comme :

$$S(t) = \sum_{i=1}^{N} E(t) exp\left(j\left(kd_{y,i}u + kd_{z,i}v\right)\right) \tag{1.31}$$

La Figure 1.11 illustre le résultat obtenu :

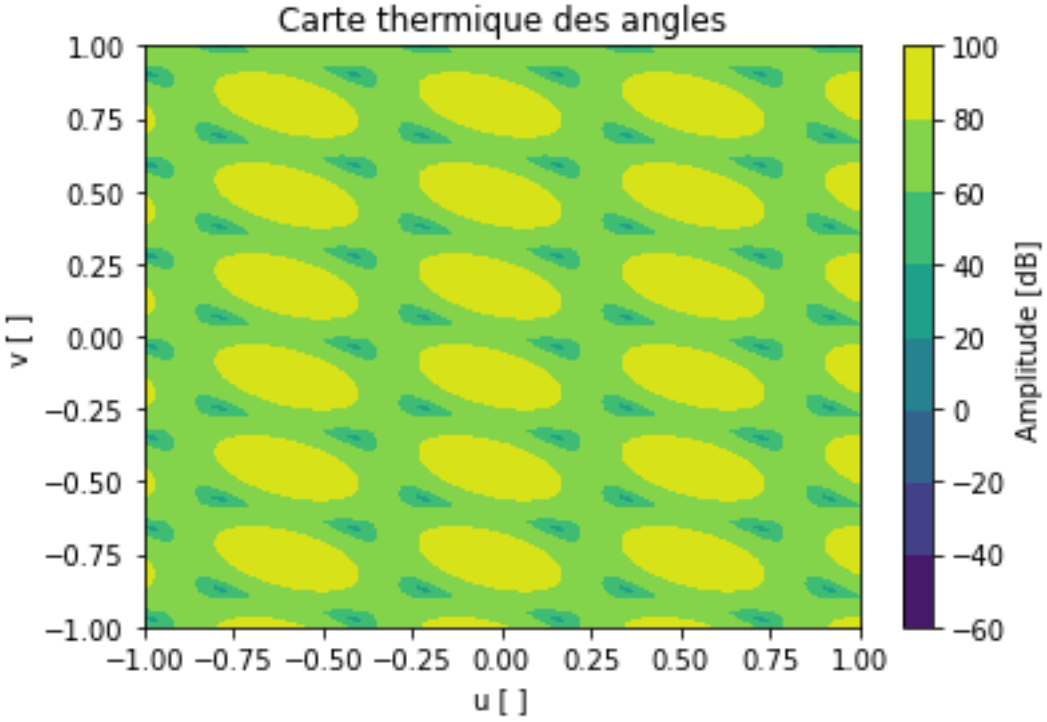


FIGURE 1.11 – La position angulaire θ, φ de la cible correspond à un des maxima locaux de cette image. Cependant, sans outil supplémentaire, il n'est pas possible de déterminer le bon maximum et donc, la position angulaire de la cible. C'est pourquoi, au Chapitre 3, nous verrons comment lever l'ambiguïté en utilisant les informations fournies par la caméra.

Chapitre 2

Simulation caméra

Dans ce travail [1], il a été décidé de modéliser un carrefour réel. Le carrefour est l'intersection entre la rue Porte Lemaître et le Boulevard Baudoin 1er à Louvain-la-Neuve. Plus précisément, il s'agit d'un carrefour à feux tel que illustré sur la Figure suivante :

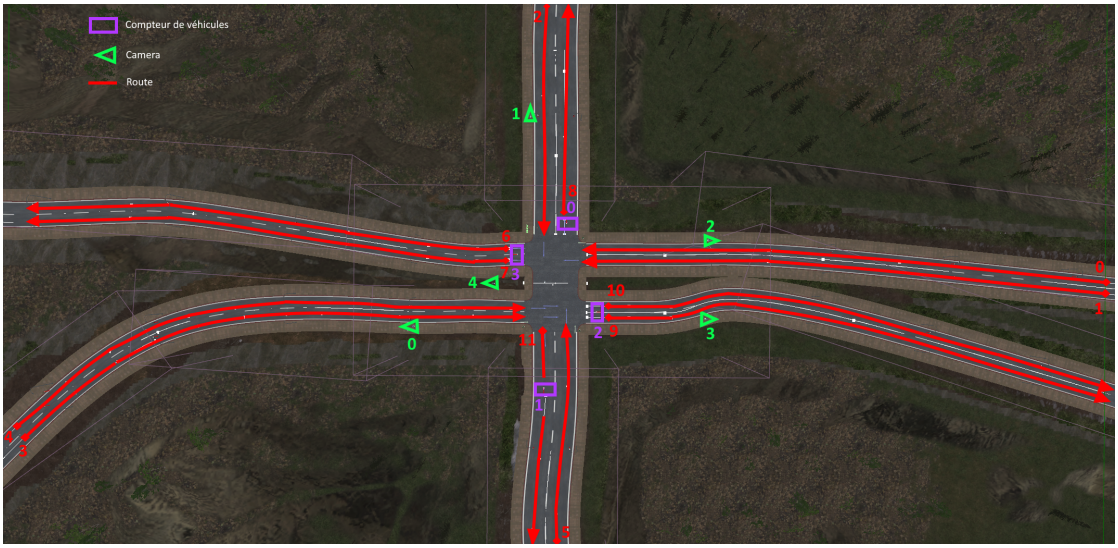


FIGURE 2.1 – Le carrefour modélisé est un carrefour à feux à 2×2 bandes dans les directions Est-Ouest et 2×1 bandes dans les directions Nord-Sud. Pour avoir une vue d'ensemble de la scène, cinq caméras (modélisées par les triangles verts) sont utilisées.

1. Il est important de rappeler que l'entièreté de la simulation a été réalisée par François-Xavier Inglese.

Ce type de carrefour est particulièrement intéressant car il est sujet à de nombreux risques d'accidents : collisions frontales dans les directions Nord-Sud, brûlage de feu rouge, céder le passage pour pouvoir changer de direction, ... De ce fait, il est intéressant d'y poser un système bimodal radar-caméra permettant d'analyser la trajectoire des différents véhicules. De plus, il est bien plus intéressant d'étudier un carrefour complexe avec de la circulation plutôt qu'une seule route avec peu de passage. Ceci permettant de mieux tester la validité et les limites des algorithmes qui seront présentés.

2.1 Information disponible

Dans cette simulation, nous avons accès à différentes informations qui nous seront utiles par la suite. On cite :

- X_{pos}, Y_{pos}, Z_{pos}
  Il s'agit de la position (x, y, z) de chaque véhicule à tout instant.
- X_{cam}, Y_{cam}, Z_{cam}
  Il s'agit de la position de chacune des caméras observant le carrefour. Ces positions sont modifiables, ce qui permet de les placer de façon optimale. Par ailleurs, les caméras sont orientables. Ceci est donné par les angles $pitch$, yaw, $roll$ qui sont également connus.
- L, l, H
  Ce sont les dimensions réelles des véhicules.
- x, y
  Ce sont les coordonnées du centre de masse de chacun des véhicules, projetées au sol.
- V_x, V_y, V_z
  Cela correspond à la vitesse instantanée de chacun des véhicules à chaque instant.

Cependant, d'autres informations sont disponibles mais non utilisées. Par exemple, les bouding boxs 2D des véhicules. Celles-ci seront calculées à l'aide de l'algorithme YOLO, tel que expliqué au Chapitre 3. La raison est que, en pratique, il est obligatoire d'utiliser un tel algorithme pour estimer ces boxs. Par ailleurs, il est également possible de modifier la luminosité (jour/nuit), ce qui permet d'analyser la robustesse des algorithmes proposés. Il est également possible de rajouter/supprimer des nuages afin de modifier un peu la luminosité de jour. Ceci ne sera pas utilisé.

2.2 Limitations de la simulation

Toute simulation admet des limitations et ne sait pas refléter complètement le cas réel. C'est pourquoi, une validation expérimentale est toujours requise afin de se confronter à la réalité. Dans cette section, nous allons expliquer les limitations de la simulation du carrefour. Celle-ci ne fonctionne que sous Windows.

2.2.1 Dynamique des véhicules

En pratique, chaque conducteur adopte son propre style de conduite. Dans cette simulation, tous les conducteurs roulent de la même façon. Par conséquent, deux véhicules roulant à la même vitesse vont parcourir exactement la même trajectoire, ce qui n'est pas le cas en pratique. En effet, pour deux conducteurs se rendant d'un point A à un point B, les trajectoires respectives seront toujours un peu différentes. Ceci est d'autant plus vrai lorsqu'il y a des tournants. Par ailleurs, une autre limitation concerne la dynamique des véhicules. En effet, pour freiner, le véhicule arrête tout simplement d'accélérer. Par conséquent, chaque véhicule finira toujours par s'arrêter mais en prenant un certain temps. Dès lors, pour éviter les accidents, les véhicules devront tous rouler à une vitesse semblable.

2.2.2 Variété des vitesses

Comme dit précédemment, le traitement des données se termine par une architecture permettant de filtrer le bruit présent dans les données. Celle-ci ayant besoin d'être entraînée, un grand nombre de données est nécessaire. Cependant, pour que le modèle puisse filtrer au mieux, il faut également que les données d'entraînement soient suffisamment différentes les unes des autres. Ceci n'est pas un problème pour les données concernant la position des véhicules étant donné que, en utilisant suffisamment de véhicules et de données, une grande variété de position sera obtenue. Ceci étant d'autant plus vrai en prenant les différents points de vue offerts par les caméras. Par contre, cela devient problématique pour les données relatives à la vitesse. En effet, la plupart des véhicules roulent à la même vitesse car, comme dit précédemment, le mode de freinage des véhicules n'est pas optimal. Il n'y a donc aucune variété possible concernant la vitesse. En conséquence, des architectures neuronales risquent de faire du sur-entraînement (*overfitting*), ce qui signifie qu'elles n'apprendront rien du tout. Ainsi, en pratique, le modèle ne fonctionnera pas du tout et ne sera pas capable de prédire correctement des vitesses. Une solution trouvée à ce problème, mais pas réaliste, est de limiter chaque bande de circulation à une certaine vitesse. Ainsi, sur des routes à deux bandes, une bande sera par exemple limitée à une vitesse de 50 [km/h] et l'autre à une

vitesse de 90 [km/h]. On se rend donc bien compte que ceci n'est pas réaliste mais que cela permet de pallier le problème de diversité de vitesse.

2.2.3 Modèle de la caméra

Les caméras utilisées dans la simulation pour observer la scène sont des caméras virtuelles. Cela signifie qu'il n'y a aucune non-idéalité dans les images qu'elles prennent. Sans entrer dans les détails, il n'y a aucune notion de flou dans l'image ce qui, comme nous le verrons par la suite, va compliquer l'extraction de l'information fournie par chacune des caméras.

2.2.4 Modélisation de l'environnement

L'environnement de la simulation est très limité. Les paysages ne sont pas très réalistes et les conditions météorologiques telles que la pluie, la neige et le brouillard ne sont pas modélisées. Les caméras sont supposées statiques et il n'y pas de vibration dûe au vent, au passage des voitures sur la route, etc. En pratique, il est clair que cela aura un effet sur la prise de données par la caméra. Finalement, la texture des voitures est trop "lisse". Il n'y pas d'aspérité notable et la lumière est donc bien réfléchie.

2.2.5 Données synthétiques

Une simulation fournit des données synthétiques. Ceci peut être un problème si elles ne sont pas assez réalistes. C'est par exemple le cas du modèle utilisé pour les caméras, comme expliqué à la section précédente. Une solution, non utilisée dans ce travail, est d'utiliser une architecture de type GAN (Generative Adversarial Network). Un tel réseau est représenté sur la Figure suivante (schéma provenant de [8]) :

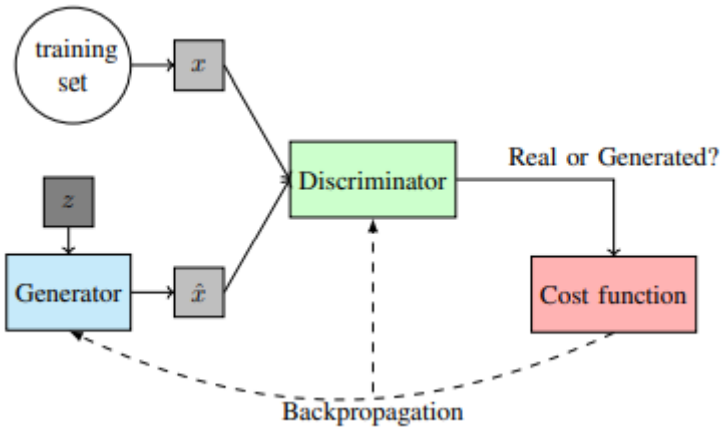


FIGURE 2.2 – Une architecture de type GAN est constituée de deux réseaux : un générateur et un discriminateur.

Une architecture GAN est constituée d'un réseau générateur et d'un réseau discriminateur. En général, les données en entrée sont des images qui sont soit réelles (x), soit synthétiques ($\hat{x}$). Sans entrer dans les détails et sortir du cadre de ce travail, le générateur consiste à générer des données synthétiques $\hat{x}$ à partir de bruit z. L'autre réseau prend en entrée une image réelle x ou synthétique $\hat{x}$ et doit déterminer si l'entrée est réelle ou non. Ainsi, une architecture GAN consiste à entraîner conjointement deux réseaux. Si le générateur fonctionne mal, les données sont trop synthétiques et le discriminateur réalise sa tâche trop aisément. Il faut donc mieux entraîner le générateur. Inversément, si le discriminateur fonctionne mal.

Dans le cadre de ce travail, nous pourrions imaginer utiliser un tel réseau pour obtenir des images plus réelles. Cependant, ceci n'a pas été réalisé mais constitue une piste d'amélioration de ce travail.

Chapitre 3

Levée des ambiguïtés du radar

Comme nous l'avons vu précédemment, une caractéristique du radar est qu'il comporte des ambiguïtés concernant l'orientation θ, ϕ de chaque cible. Ainsi, il n'est pas possible, sans outil supplémentaire, de déterminer la position (x, y, z) d'une cible à l'aide des données mesurées par cet instrument. Une façon de lever les ambiguïtés est d'utiliser les données mesurées par la caméra. Ainsi, la première étape consiste à estimer la position θ, ϕ de chaque cible avec la caméra.

Une caméra est un instrument qui permet de fournir une représentation 2D du monde physique. Autrement dit, elle réalise une transformation du 3D vers le 2D. Cette transformation, notée h, associe à toute position (x, y, z) d'une cible des coordonnées (u, v) dans l'image.

$$\begin{array}{rccc} h & : & R^3 & \rightarrow & R^2 \\ & & (x, y, z) & \mapsto & (u, v) \end{array} \tag{3.1}$$

En conséquence de cette transformation, le passage vers une dimension inférieure entraîne une perte d'information. Celle-ci correspond à la profondeur que nous ne pouvons pas directement retrouver à partir des coordonnées (u, v). Ainsi, réaliser la transformation inverse h^{-1}, consistant à passer d'une image 2D vers le monde 3D, n'est pas une chose aisée puisqu'il faut estimer cette profondeur. Plusieurs techniques sont possibles et seront expliquées à la Section 3.2.3 :

1. Utilisation des coordonnées homogènes ;
2. Utilisation de deux caméras ;
3. Utilisation de l'angle de vue de la caméra.

Les deux premières méthodes fournissent des résultats assez précis mais requièrent l'utilisation de la distance focale de la caméra. Comme précisé au Chapitre 2, la caméra simulée est parfaite : elle ne réalise pas un focus à une certaine profondeur et donc, il n'y a pas de flou dans l'image. Cette caméra virtuelle n'a donc pas

de distance focale mais uniquement un angle de vue. C'est pourquoi l'utilisation de la troisième méthode a été choisie. Néanmoins, elle fournit une estimation de la profondeur qui comporte parfois des erreurs grossières qu'il faudra minimiser tant que possible.

Pour déterminer la position et la vitesse d'une cible à partir d'une image de la caméra, le schéma suivant est utilisé :

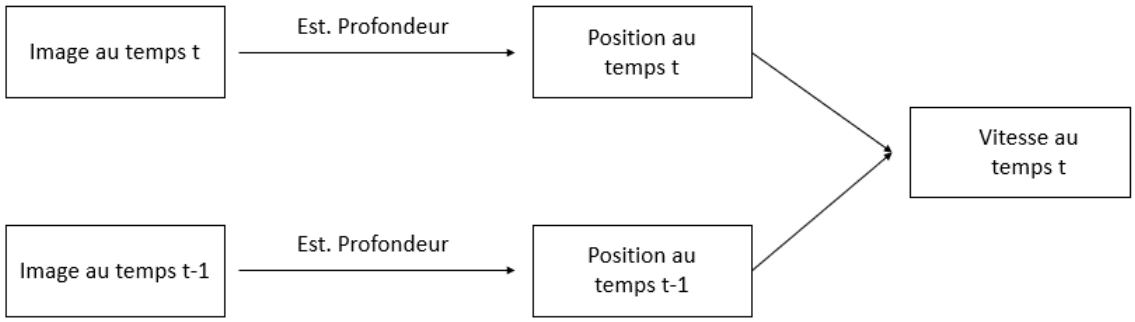


FIGURE 3.1 – Schéma bloc permettant d'estimer la position et la vitesse d'une cible. Pour ce faire, la position à l'instant t d'un véhicule nécessite une estimation de la profondeur présente sur l'image. La vitesse d'un véhicule à l'instant t, quant à elle, nécessite la position aux instants t et $t-1$.

Le schéma ci-dessus illustre la façon dont la position et la vitesse d'un véhicule peuvent être estimées à partir des données de la caméra. On y remarque deux choses. La première est que, pour estimer la profondeur de la cible, et donc sa position, il est nécessaire de connaître au préalable où elle se situe dans l'image. D'un autre côté, une conséquence importante est que l'estimation de la vitesse du véhicule sera conditionnée par l'estimation de la position de ce même véhicule. En effet, une estimation instable de la position entraînera une estimation erronée de la vitesse. Ceci fera l'objet des sections suivantes.

3.1 You Only Look Once (YOLO)

L'algorithme You Only Look Once (YOLO) permet de faire de la détection d'objets à l'instar des véhicules, des piétons, ... Pour ce faire, il fournit une bounding box autour des objets, une classe pour l'objet ainsi que la confiance attribuée à cette classe. Cet algorithme, présenté pour la première fois en 2015, en est aujourd'hui

à sa cinquième version. Sur Internet, il est possible d'en trouver de nombreuses implémentations open-sources ([9] ou plus récemment [10]).

3.1.1 Principe de fonctionnement

L'algorithme YOLO est un réseau neuronal de type CNN (Convolutional Neural Network) permettant de faire de la détection et de la classification d'objets. Ainsi, non seulement il détermine la position d'un objet sur l'image mais il lui fournit également une classe avec une certaine probabilité. Sans entrer dans les détails, une brève explication de son fonctionnement est fournie dans cette section. Ceci se base sur [11], dont les images sont tirées.

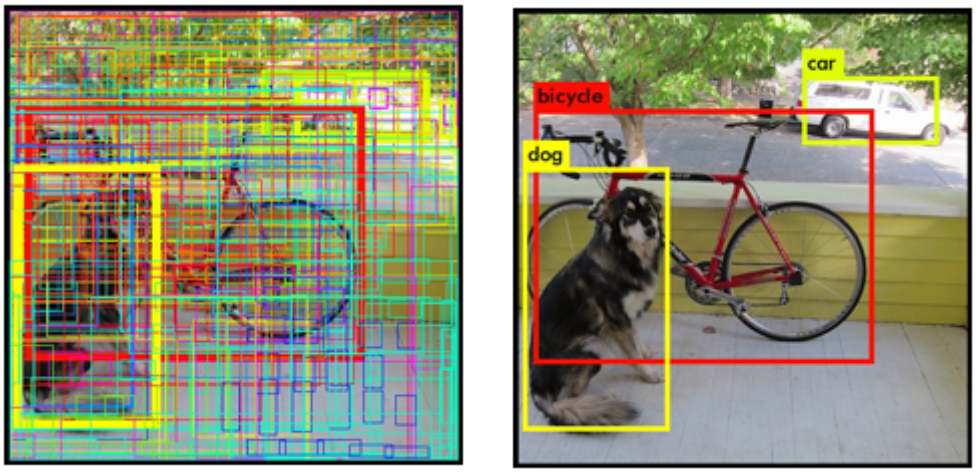
Globalement, le réseau prend en entrée une image et la divise en une grille de cellules. Chacune de ces cellules va alors prédire B bounding boxs auxquelles un score de confiance est associé. Celui-ci exprime de combien l'algorithme est sûr que la box prédite entoure bel et bien un objet. Ces deux étapes sont illustrées sur les deux Figures suivantes :

(a) Cette Figure illustre le fonctionnement de l'algorithme YOLO : à partir d'une image, il la divise en une grille de cellules.

(b) Pour chaque cellule, l'algorithme prédit plusieurs bounding boxs. Au plus YOLO est certain que celles-ci entourent un objet, au plus elles sont épaisses.

FIGURE 3.2 – Ces deux Figures illustrent les deux premières étapes de l'algorithme YOLO.

Dans un second temps, chaque cellule prédit une classe pour chacune des boxs. L'algorithme va alors combiner le score de confiance et la classe prédite en un seul score. Celui-ci indique de combien l'algorithme est sûr qu'une box contient une classe d'objet en particulier. Ainsi, au moyen de ces trois étapes, des centaines de boxs sont prédites auxquelles est associé un score de confiance dont la valeur oscille entre 0 et 1. En général, si le réseau est bien entraîné, les boxs entourant les objets les plus intéressants à détecter sur l'image (le chien, le vélo et la voiture sur la Figure 3.2) ont un score élevé. Les autres ont un score plutôt faible. Puisqu'elles ne sont pas intéressantes, l'utilisateur peut ne pas en tenir compte en fixant un seuil. Dès lors, seuls les scores plus élevés à ce seuil seront gardés. Ces deux étapes finales sont illustrées aux Figures suivantes :



(a) L'algorithme combine les boxs et les classes prédites. Pour illustrer cela, chaque classe correspond à une couleur bien précise. On constate qu'il y a énormément de boxs prédites.

(b) L'utilisateur, en utilisant un seuil, peut ne garder que les prédictions les plus fiables. Si le réseau est bien entraîné, il s'agit des objets les plus intéressants à détecter sur l'image.

FIGURE 3.3 – Ces deux Figures illustrent les deux dernières étapes de l'algorithme YOLO.

3.1.2 Résultats avec YOLOV5

Il a été décidé d'utiliser YOLOV5 ([10]) étant donné qu'il s'agissait de la version la plus récente de l'algorithme. Les performances des différents modèles possibles de cette version sont représentées sur la Figure 3.4. Tous ces modèles ont été pré-entraînés sur l'ensemble de données COCO (Common Objects in Context [12]).

Etant donné que les détections de véhicules étaient bien souvent correctes, il a été décidé de ne pas entraîner les modèles. De plus, ceci aurait principalement permis d'avoir un meilleur intervalle de confiance, ce qui n'est pas utilisé dans le cadre du travail.

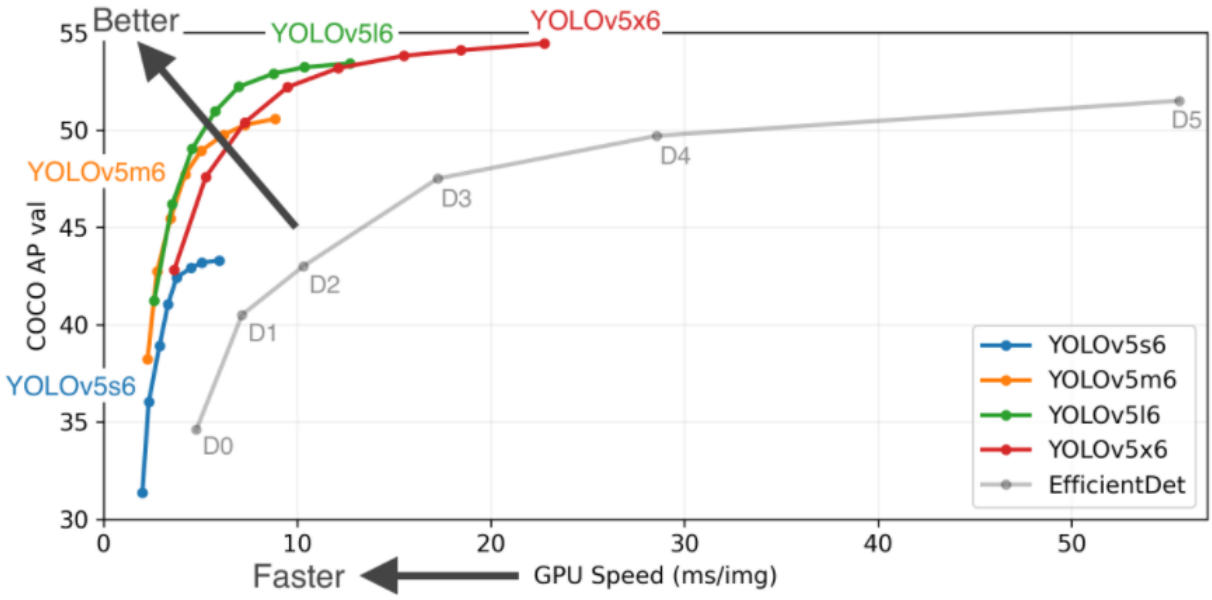


FIGURE 3.4 – Cette Figure représente les performances des différents modèles, évaluées selon la métrique *Mean Average Precision*. On constate que le modèle YOLOV5s est le plus rapide mais le moins performant. A l'inverse, le modèle YOLOV5x fournit les meilleurs résultats mais prend le plus de temps. Cette Figure provient de [10].

Etant donné qu'une implémentation en temps-réel n'est pas le but de ce travail, le critère de choix le plus important concerne les performances du modèle. La Figure ci-dessus illustre les performances de quatre versions de YOLOv5 entraînées sur l'ensemble de données COCO. Celui-ci fournit des images d'objets que nous pouvons rencontrer dans la vie de tous les jours ainsi que la classe associée à ces objets. Ceci justifie donc le fait qu'il n'est pas nécessaire d'entraîner à nouveau l'algorithme YOLO. Par contre, s'il fallait déterminer le type de voitures (Audi, Peugeot, ...) présentes sur une image, il faudrait dans ce cas entraîner le réseau sur un autre ensemble de données. Sur la Figure 3.4, on constate que la meilleure précision est atteinte avec YOLOv5x. Par conséquent, nous utiliserons uniquement ce modèle, bien qu'il soit également le plus lent. Par ailleurs, ce modèle est utilisé en tant que boîte noire et nous prenons pour acquise son implémentation. Nous pouvons cependant jouer sur un paramètre : le seuil qui délimite la confiance

minimale que l'on souhaite avoir pour une détection d'objets. Cette valeur varie entre 0 et 1. Une valeur très faible, comme par exemple 0.1, aura pour conséquence que l'algorithme détectera énormément de cibles. Etant donné que l'objectif du travail n'est pas de détecter tous les objets présents sur une image mais uniquement les cibles mouvantes telles que les voitures et les camions, il n'est pas judicieux de travailler avec une valeur de seuil faible. A l'inverse, si celle-ci est élevée, comme par exemple 0.8, alors l'algorithme risque de ne pas considérer certains objets. L'impact du seuil est illustré à la Figure 3.3.

Toutefois, une valeur de seuil trop élevée est plus contraignante qu'une valeur trop faible. En effet, il vaut mieux détecter trop d'objets et ensuite de faire le tri, que d'en détecter trop peu. Un objet non détecté ne pouvant plus être récupéré par la suite. Ceci est d'autant plus vrai la nuit : l'algorithme YOLO a plus de mal à détecter des véhicules et donc, fournit les détections avec une confiance plus faible. C'est pourquoi, il a été décidé d'utiliser un seuil de 0.25, ce qui semble fournir un bon compromis. Nous obtenons ce genre de résultats :

FIGURE 3.5 – Cette Figure illustre la détection d'objets réalisée par l'algorithme YOLO avec un seuil de 0.25. On constate qu'il est capable de détecter correctement des voitures mais également d'autres objets comme des personnes et des feux de signalisation. Nous pouvons constater que, malgré le fait que nous n'ayons pas entraîné le modèle, celui-ci fournit tout de même les résultats escomptés. Par ailleurs, on constate également que le feu de signalisation situé sur la partie verticale du poteau n'est pas détecté, ce qui résulte de la valeur du seuil choisi.

Comme illustré sur la Figure 3.5, la détection d'objets avec l'algorithme YOLO est fructueuse. Cependant, elle nous donne des informations inutiles : il n'est pas nécessaire de détecter les feux de signalisation ainsi que les personnes présentes dans les véhicules. Ainsi, avant de traiter les données, il est nécessaire de supprimer ces données superflues. Ceci est fait en regardant les classes des véhicules et en rejetant celles relatives aux personnes et aux feux de signalisation. Toutefois, ceci pourrait être imprudent : il y a en effet une probabilité non nulle que l'algorithme considère une voiture comme étant une personne. De façon générale, toutes les voitures ont toujours bien été détectées par l'algorithme ce qui signifie que nous pouvons supprimer ces deux catégories d'objets. Ceci peut s'expliquer par le fait que l'algorithme YOLO est pré-entraîné sur l'ensemble de données COCO ([12]) qui contient plusieurs dizaines de catégories d'objets. Ainsi, en rejetant uniquement deux classes (celles des personnes et des feux de signalisation), la probabilité est très faible de ne pas détecter une voiture. De plus, intuitivement, pour que l'algorithme se trompe et confonde une voiture avec une personne, il faudrait que l'aire de la bounding box soit très petite. Ceci correspondrait au cas où la voiture est très loin de la caméra, ce qui n'est pas le cas le plus intéressant étant donné que le radar ne captera probablement pas ce véhicule [1]. Par ailleurs, une autre non-idéalité de la détection d'objets faîte par l'algorithme YOLO est illustrée à la Figure suivante :

FIGURE 3.6 – Sur cette Figure, on y voit deux camions et une voiture. Nous pouvons constater que le camion le plus proche de la caméra (celui de gauche) est à la fois détecté en tant que camion mais également en tant que train. Il en résulte qu'il y a deux détections pour une seule cible.

1. Pour rappel, le radar peut capter des véhicules jusqu'à une distance maximale de $70.122[m]$. A titre d'exemple, la voiture brune (sur la Figure 3.5) en train de tourner dans le carrefour se trouve à une distance d'environ $60[m]$, ce qui n'est pas loin de la distance maximale du radar. Par conséquent, on peut supposer que rejeter les classes des personnes et des feux de signalisation ne rejettera pas une voiture. Ceci est d'autant plus vrai lorsqu'il y a beaucoup de luminosité.

La non-idéalité illustrée à la Figure 3.6 devient problématique lors de l'association des données du radar avec celles de la caméra. En effet, le radar ne captera normalement qu'une seule fois le camion et donc, on risque de se tromper lors de l'étape d'association des données. Par conséquent, cette non-idéalité doit-être résolue. Pour ce faire, sur une image, on considère le centre de chacune des bounding boxs. Si la distance entre deux centres est inférieure à un certain seuil, alors il s'agit du même véhicule. La valeur du seuil ne doit pas être trop grande ni trop petite. En effet, une valeur trop grande risque de considérer des doublons lorsqu'il n'y en a pas. Par exemple, sur la Figure 3.6, la voiture et le camion situés dans le coin supérieur droit de l'image ont des bounding boxs très proches. Par conséquent, la distance entre le centre de chaque box est petite. Une grande valeur de seuil implique donc que l'algorithme de recherche des doublons va considérer que la voiture et le camion correspondent à un seul et même véhicule, ce qui est faux ! A contrario, une valeur de seuil trop petite risque de ne pas supprimer tous les doublons. Par exemple, sur la Figure 3.6, le camion de gauche est considéré à la fois comme un camion mais également comme un train. Dès lors, il y a bien un doublon à supprimer. La distance entre le centre des deux boxs vaut environ 9 pixels. Dès lors, un seuil trop petit ne permettra pas de supprimer ce doublon.

Dans ce travail, un bon compromis semble être une valeur de 20 ou 25 pixels [2]. Finalement, il est à noter que l'algorithme de suppression des doublons va supprimer un seul des deux doublons, sans aucune préférence. En effet, la classe des objets détectés n'est pas utilisée. Cela a donc peu d'importance de considérer l'un ou l'autre objet. Cependant, s'il était nécessaire d'utiliser la classe des objets, le mieux serait de supprimer la détection à laquelle YOLO attribue le moins de confiance.

3.2 Extraction de la distance

Une image est une représentation 2D du monde. Ainsi, lors du passage du monde 3D vers le monde des pixels, une information importante est perdue : la profondeur. Par conséquent, celle-ci doit être estimée pour être en mesure d'estimer la distance entre un véhicule et la caméra.

3.2.1 Pinhole camera model

Dans un premier temps, il faut modéliser la caméra. Ceci est fait en utilisant le modèle simple et largement utilisé du *Pinhole Camera model*, sténopé en français. Dans ce modèle, la caméra correspond à une boîte, n'a pas de lentille et toute la

2. Cette valeur dépend évidemment des dimensions des images considérées. Dans ce travail, elles sont de dimensions 1920×1080.

lumière passe à travers un petit trou appelé *pinhole*. Par l'effet *camera obscura*, la scène observée par la caméra est projetée sur l'arrière de la boîte et est inversée. Ceci est illustré à la Figure suivante :

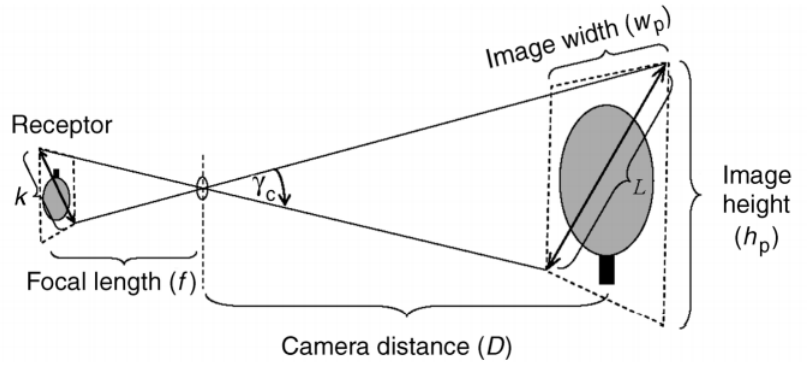


FIGURE 3.7 – Cette Figure représente l'observation d'une forme par une caméra de type *Pinhole*. Nous pouvons observer l'effet *camera obscura*. Ce schéma provient de [13].

Comme pour tout modèle mathématique, le *Pinhole Camera model* ne modèlise pas parfaitement une caméra. En effet, il ne tient pas compte de la distorsion provoquée par la lentille, ni du niveau de flou sur les objets où la caméra ne se concentre pas. Néanmoins, comme nous le verrons par la suite, il constitue une bonne approximation.

Par ailleurs, ce modèle permet d'exprimer les coordonnées (u, v) du plan de l'image par rapport aux coordonnées 3D (x, y, z). Pour ce faire, utilisons les notations de la Figure suivante provenant de [14] :

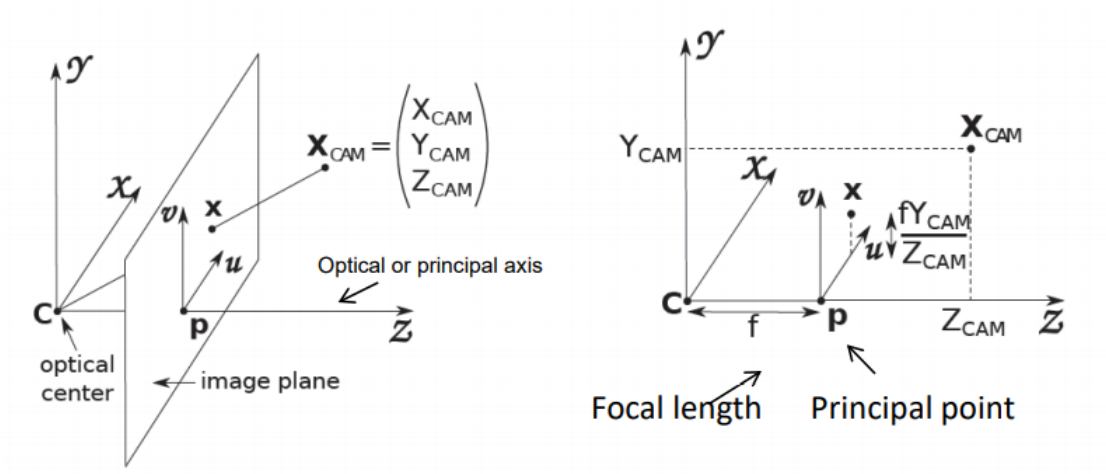


FIGURE 3.8 – Sur cette Figure, on considère un point 3D X_{CAM} dont les coordonnées cartésiennes sont $(X_{CAM}, Y_{CAM}, Z_{CAM})$. Le centre optique de la caméra est noté C et le point principal, correspondant au point d'intersection entre l'axe OZ et le plan de l'image, est noté P.

Selon [14], étant donné la définition du point C et du point P, on en déduit directement que la distance focale f de la caméra correspond à la distance entre ces deux points. En utilisant des triangles semblables, on peut directement exprimer les coordonnées (u, v) de l'image en fonction des coordonnées cartésiennes (x, y, z). Ceci est exprimé par les deux relations suivantes :

$$\begin{cases} u = \frac{fX_{CAM}}{Z_{CAM}} \\ v = \frac{fY_{CAM}}{Z_{CAM}} \end{cases} \tag{3.2}$$

Ainsi, le modèle du *pinhole* nous permet d'affiner la transformation donnée par l'Equation (3.1). Celle-ci devient :

$$\begin{array}{rccc} h & : & R^3 & \rightarrow & R^2 \\ & & (x, y, z) & \mapsto & (u, v) = \left(\frac{fX_{CAM}}{Z_{CAM}}, \frac{fY_{CAM}}{Z_{CAM}}\right) \end{array} \tag{3.3}$$

3.2.2 Estimation du centre de masse du véhicule

Le centre de masse d'un véhicule se définit comme le point où s'appliquent les effets d'inertie. Autrement dit, c'est le point de l'espace de référence que l'on considère comme le point d'application du vecteur vitesse du véhicule. De façon rigoureuse, un véhicule est constitué d'un ensemble de points M_i de coordonnées cartésiennes (x_i, y_i, z_i) de masse volumique $\rho(M_i)$. Dans ce cas, [15] définit le centre

de masse G de coordonnées cartésiennes (X_G, Y_G, Z_G) :

$$\begin{cases} X_G = \frac{\int \rho(M) x dV}{m} \\ Y_G = \frac{\int \rho(M) y dV}{m} \\ Z_G = \frac{\int \rho(M) z dV}{m} \end{cases} \tag{3.4}$$

Avec $m = \int \rho(M) dV$.

Cependant, déterminer le centre de masse d'un véhicule en utilisant l'Equation (3.4) devient vite compliqué. De plus, étant donné que nous travaillons dans le monde de la caméra avec des coordonnées 2D et que nous n'avons en pratique pas accès à la masse volumique des matériaux des véhicules, nous devons approximer le calcul du centre de masse. Pour ce faire, en estimant que ce point de référence se situe au centre du véhicule, nous pouvons estimer que, dans le monde des pixels, il correspond au centre de la bounding box donnée par l'algorithme YOLO. Cependant, ceci n'est pas tout à fait vrai. En effet, dans la plupart des voitures, le moteur se trouve à l'avant, ce qui a pour effet que le centre de masse aura plutôt tendance à se retrouver sur l'avant du véhicule. Par ailleurs, comme dit précédemment, lors du passage du monde 3D vers le monde 2D, nous perdons la notion de profondeur. Ceci aura un impact important lors du calcul de la distance entre le véhicule et la caméra. Ceci sera discuté à la Section 3.2.3. Enfin, il est important de noter que l'estimation du centre de masse du véhicule est entièrement dépendante de la précision de l'algorithme YOLO. Ceci est illustré à la Figure suivante :

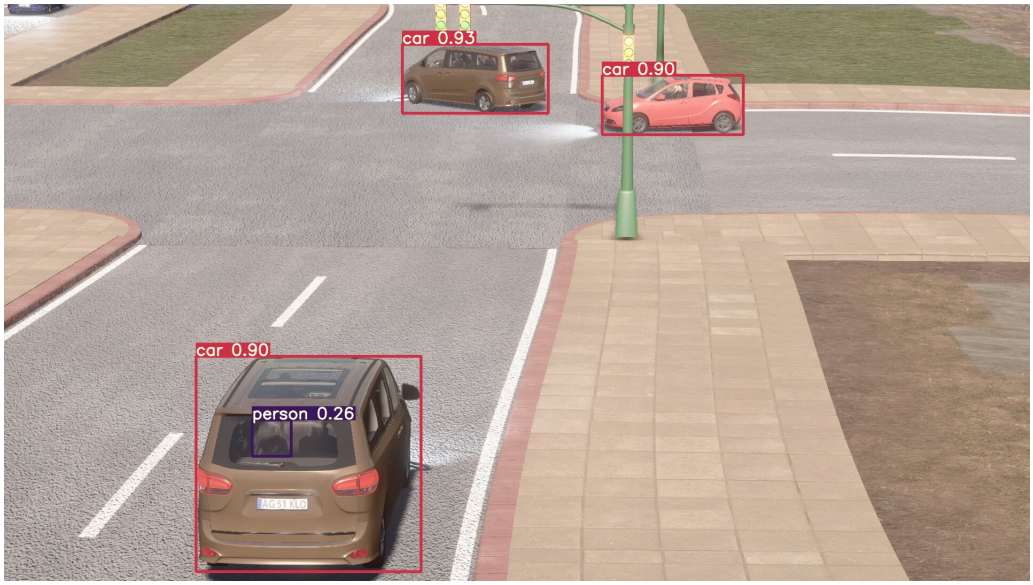


FIGURE 3.9 – Comme nous pouvons le constater, la bounding box de la voiture rose épouse parfaitement les contours du véhicule. Par contre, les bounding boxs des deux autres voitures n'épousent pas parfaitement tous les contours. Ceci est dû à l'orientation des véhicules par rapport à la caméra.

Comme nous pouvons le constater sur la Figure ci-dessus, les véhicules vus de côtés auront une meilleure estimation concernant le centre de masse. Par contre, les véhicules vus de derrière (ou de face) ainsi que ceux en train de tourner vont avoir tendance à décaler la position du centre de masse de quelques pixels. Comme nous allons le voir, ceci va avoir pour effet de fausser la distance objet-caméra.

3.2.3 Estimation de la distance objet-caméra

Dans la littérature, nous pouvons trouver bien des manières de calculer la distance séparant un objet (un véhicule, une personne, ...) de la caméra. Dans cette section, nous allons présenter le fonctionnement de trois méthodes couramment utilisées, bien que nous n'en ayons utilisé qu'une seule. La raison de ce choix est dû au fait que ces méthodes utilisent en général des caméras réelles pour lesquelles la connaissance de paramètres tels que la taille du senseur ou de la distance focale est connue. Dans le cas de la simulation de la caméra, ces paramètres ne sont pas disponibles étant donné qu'il s'agit de la simulation d'une caméra virtuelle.

Coordonnées homogènes

Les coordonnées homogènes permettent d'exprimer plus facilement la transformation de coordonnées 3D vers des coordonnées 2D du monde des images, telle qu'exprimée par l'Equation (3.3). En effet, comme nous allons le voir, cette transformation s'exprime à l'aide d'une écriture matricielle et est donc linéaire. Dans ce qui suit, la distance focale de la caméra est notée f et le raisonnement se base sur ce qui a été fait dans [14].

Pour passer dans le monde des coordonnées homogènes, on ajoute une nouvelle dimension par rapport à celle de l'espace de projection. La valeur associée à cette nouvelle dimension est telle que, en divisant la valeur de chaque autre dimension par celle-ci, on obtient les valeurs associées à chacune des coordonnées non homogènes. En utilisant le modèle du *pinhole*, on obtient le système suivant :

$$\begin{pmatrix} fX_{CAM} \\ fY_{CAM} \\ Z_{CAM} \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X_{CAM} \\ Y_{CAM} \\ Z_{CAM} \\ 1 \end{pmatrix} \tag{3.5}$$

De façon plus compacte, ce système peut se réécrire par :

$$Y = CX_{CAM} \tag{3.6}$$

Dans cette nouvelle équation, la matrice C s'appelle la matrice caméra. Nous pouvons remarquer que, connaissant la distance focale f de la caméra, cette matrice est entièrement déterminée. Cette équation nous permet donc de réaliser la projection du monde 3D vers le monde des pixels de façon linéaire. De plus, on peut remarquer qu'il n'y a pas qu'une seule façon de définir les coordonnées homogènes. En effet, seul le rapport entre les valeurs de chaque dimension et celle de la dimension ajoutée importe.

Par ailleurs, il est courant de généraliser le modèle (3.5). Pour ce faire, il suffit de modifier la définition de la matrice caméra en ajoutant des paramètres en plus de la distance focale. Ceux-ci sont regroupés en deux catégories : les paramètres intrinsèques et les paramètres extrinsèques. Nous pouvons tenir compte de ces paramètres en multipliant la matrice caméra C par la matrice intrinsèque et la matrice extrinsèque.

La matrice intrinsèque est de dimensions 3×3 et est définie de la façon suivante :

$$K = \begin{pmatrix} k_u & S_{uv} & c_u \\ 0 & k_v & c_v \\ 0 & 0 & 1 \end{pmatrix} \tag{3.7}$$

Nous pouvons constater que cette matrice apporte donc cinq degrés de liberté, dont les significations sont les suivantes :

— k_u et k_v
  Ce sont les facteurs d'agrandissement de l'image. Ils ont en général une valeur similaire et expriment le fait que les pixels ne sont pas toujours carrés. Ainsi, dans le cas idéal où les pixels sont carrés, ces deux valeurs sont identiques. Par ailleurs, on définit souvent f_x et f_y, les distances focales exprimées en largeur et hauteur de pixel, respectivement par $f_x = k_u f$ et $f_y = k_v f$.

— c_u et c_v
  Ce sont les coordonnées du point principal exprimées dans le plan de l'image. Elles correspondent donc à la translation à réaliser pour passer du point principal de la caméra à une nouvelle origine dans le plan de l'image (en général le coin supérieur gauche).

— S_{uv}
  Ce paramètre exprime la non-orthogonalité entre les lignes et les colonnes de pixels. En général, cet effet indésirable peut être négligé, ce qui consiste simplement à mettre ce paramètre à zéro.

Dans l'équation (3.6), les coordonnées du point X_{CAM} sont exprimées dans le repère de la caméra. Ainsi, ce point correspond à un point X dont les coordonnées non homogènes sont exprimées dans un repère arbitraire que l'on considère comme l'origine. En pratique, on a donc plutôt accès au point X et non au point X_{CAM}. Il faut donc réaliser une rotation et une translation afin de passer du repère arbitraire au repère de la caméra. Ceci est exprimé par la matrice extrinsèque.

Un point dans l'espace ayant au maximum six degrés de liberté, la matrice extrinsèque demande donc d'estimer six paramètres supplémentaires : 3 pour la rotation et 3 pour la translation. Par le théorème des rotations d'Euler, le déplacement d'un point peut être représenté par trois rotations, chacune autour d'un des trois axes du repère arbitraire, suivi d'une translation. Ceci est exprimé par l'équation suivante :

$$X_{CAM} = R(X - T) \tag{3.8}$$

Où $R = R_z R_y R_x$ est la matrice de rotation et $T = (T_x, T_y, T_z)$ est la translation.

Dans l'équation ci-dessus, les angles de rotation des rotations R_x, R_y et R_z correspondent aux angles d'Euler, respectivement *yaw*, *pitch*, *roll*. Ces trois rotations sont alors définies de la façon suivante :

$$R_x(\gamma_x) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma_x) & -\sin(\gamma_x) \\ 0 & \sin(\gamma_x) & \cos(\gamma_x) \end{pmatrix} \tag{3.9}$$

Où γ_x est l'angle *yaw*.

$$R_y(\gamma_y) = \begin{pmatrix} \cos(\gamma_y) & 0 & \sin(\gamma_y) \\ 0 & 1 & 0 \\ -\sin(\gamma_y) & 0 & \cos(\gamma_y) \end{pmatrix} \tag{3.10}$$

Où γ_y est l'angle *pitch*.

$$R_z(\gamma_z) = \begin{pmatrix} \cos(\gamma_z) & -\sin(\gamma_z) & 0 \\ \sin(\gamma_z) & \cos(\gamma_z) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.11}$$

Où l'angle γ_z est l'angle *roll*.

Ainsi, au vu de ce qui précède concernant les matrices intrinsèque et extrinsèque, nous pouvons définir la nouvelle matrice caméra de la façon suivante :

$$P = \begin{pmatrix} K & 0 \end{pmatrix} C \begin{pmatrix} R & T \\ 0 & 1 \end{pmatrix} \tag{3.12}$$

$$= KC \begin{pmatrix} R & T \end{pmatrix} \tag{3.13}$$

Par conséquent, l'Equation (3.6) devient :

$$Y = PX_{CAM} \tag{3.14}$$

Ainsi, dans le cas général, le modèle du sténopé contient 11 degrés de liberté supplémentaires par rapport au cas de base : cinq provenant des paramètres intrinsèques et six provenant des paramètres extrinsèques. En pratique, ces paramètres n'étant pas connus, il faut les estimer. Pour ce faire, il faut connaître au préalable des correspondances 2D-3D. Etant donné que ces correspondances fournissent chacune deux équations, il faut au minimum six correspondances différentes pour être capable d'estimer ces paramètres. Evidemment, avoir plus de correspondances permet d'affiner cette estimation. Ceci peut être réalisé par l'algorithme de Levenberg-Marquardt. Pour le lecteur intéressé, la calibration de la caméra qui sera utilisée

pour la validation expérimentale et qui se base sur le raisonnement précédent a été effectuée dans [16].

Finalement, une fois que la matrice caméra P de l'Equation (3.14) est estimée, il est facile de transformer des coordonnées cartésiennes arbitraires en pixels. Dans notre cas, nous souhaitons réaliser le cheminement inverse : passer du monde 2D vers le monde 3D. Pour ce faire, dans un premier temps, il faut inverser la matrice caméra. Etant donné que celle-ci est de dimensions 3×4, ceci est fait par une méthode d'inversion pseudo-inverse. Par ailleurs, grâce à la géométrie épipolaire, nous savons qu'à un pixel de coordonnées (u, v) correspond une ligne de points possibles en coordonnées cartésiennes (x, y, z). Ceci est tout à fait normal étant donné que nous passons vers une plus grande dimension. Il faut donc fournir une information supplémentaire qui est la profondeur. Nous avons alors tout en main pour réaliser la conversion des pixels vers les coordonnées 3D. Enfin, il est bon de noter que la librairie OpenCV de Python fournit de bonnes méthodes permettant d'implémenter la transformation en coordonnées homogènes.

En outre, il est possible de considérer des modèles de caméra tenant compte de la distorsion de la lentille. Néanmoins, étant donné qu'une grosse partie de ce travail se base sur une simulation parfaite d'une caméra, cela ne sera pas expliqué dans cette section.

Estimation de la profondeur à l'aide de deux caméras

A la section précédente, nous travaillions en monovision : nous avions accès uniquement à une seule caméra et nous cherchions à déterminer la position 3D d'un objet à partir de ses coordonnées dans le plan de l'image. Dans cette section, nous considérons le cas où nous avons accès à deux caméras et donc, à deux vues différentes d'une même scène. Nous sommes dans le cas de la stéréo-vision. Le raisonnement suivant se base sur ce qui a été effectué dans [14].

Pour simplifier les calculs, on considère deux caméras identiques et alignées le long de l'axe OX, telles que représentées sur la Figure suivante :

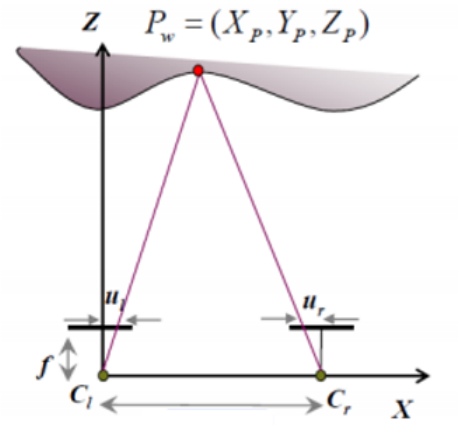


FIGURE 3.10 – Ce schéma illustre deux caméras, notées par C_l et C_r, observant un même point P_w. On cherche à estimer la profondeur Z_P. Les variables u_l et u_r représentent la localisation du point P_W dans le plan de l'image.

On note b la distance entre les deux caméras, supposée connue. De plus, la distance focale des caméras est notée f et est également supposée connue. En utilisant des triangles semblables, on a :

$$\frac{f}{Z_P} = \frac{u_l}{X_P} \tag{3.15}$$

$$= \frac{-u_r}{b - X_P} \tag{3.16}$$

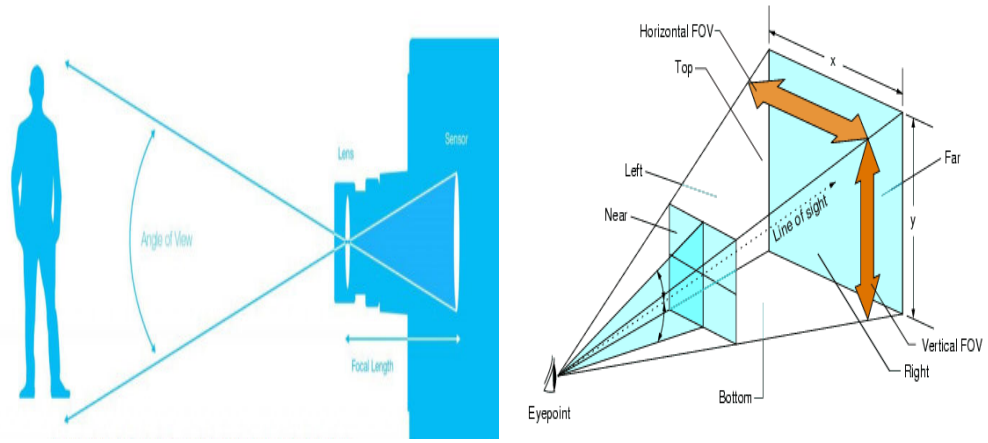
En isolant Z_P, une estimation de la profondeur est donnée par :

$$Z_P = \frac{bf}{u_l - u_r} \tag{3.17}$$

Dans cette équation, la différence $u_l - u_r$ s'appelle la disparité et correspond à la différence de localisation du point P_W dans le plan de l'image de chacune des caméras. Sans entrer dans les détails, on peut constater que la distance b entre les deux caméras joue un rôle fondamental. En effet, si elle est trop petite, les erreurs d'estimation de la profondeur seront grandes car la triangulation se fera avec des lignes presque parallèles. Inversément, si la distance est trop grande, il sera plus compliqué de faire une correspondance entre les deux plans des caméras.

Utilisation de l'angle de vue de la caméra

Une caméra bénéficie, entre autre, de deux caractéristiques importantes : son angle de vue et son champ de vue. La première correspond à l'angle maximal avec lequel la lentille de la caméra peut voir. La seconde correspond à ce que la caméra, avec la lentille, va réellement voir. Ces deux caractéristiques sont bien souvent exprimées en degrés et peuvent être mesurées horizontalement, verticalement ou encore, en diagonal. Elles sont toutes les deux illustrées aux Figures ci-dessous :



(a) (b)

FIGURE 3.11 – A gauche, la Figure illustre l'angle de vue d'une caméra mesuré verticalement. On voit qu'il s'agit de l'angle maximal avec lequel la lentille peut observer une scène. A droite, la Figure illustre le champ de vue de la caméra. Ces illustrations proviennent de [17] et [18].

Le champ de vue d'une caméra peut être calculé à l'aide de la distance focale de la caméra et d'une des dimensions du senseur. On note W la dimension horizontale et H la dimension verticale. La distance focale est notée f. L'angle de vue horizontal (HAOV), vertical (VAOV) et diagonal (DAOV) sont alors définis de la manière suivante :

$$HAOV = 2 \arctan \frac{W}{2f} \tag{3.18}$$

$$VAOV = 2 \arctan \frac{H}{2f} \tag{3.19}$$

$$DAOV = 2 \arctan \frac{\sqrt{W^2 + H^2}}{2f} \tag{3.20}$$

Dans le cadre de ce travail, l'angle de vue de la caméra a été imposé, tant pour la partie simulation que pour la partie validation expérimentale. En effet, nous

avons repris la caméra utilisée par l'ancien mémorant Alexis Duflot et nous avons considéré cet angle de vue, à savoir 28°, pour les simulations. Ceci nous permettra, par la suite, de comparer les résultats entre un dispositif bimodal simulé et un dispositif bimodal réel. Par ailleurs, il est évident que le choix de cet angle est crucial et dépend de ce qui est souhaité. En effet, un grand angle de vue permettra d'avoir un champ de vue plus grand. Ainsi, pour une même scène, la caméra permettra d'observer plus de choses. Ceci peut être particulièrement intéressant dans le cas d'une caméra de surveillance. A l'inverse, un angle de vue plus petit aura un champ de vue plus petit. Cependant, pour une même taille d'image, les objets occuperont une plus grande place et donc, plus de détails et d'information concernant ceux-ci seront disponibles. Nous pouvons donc constater qu'il y a un certain compromis. Dans le cadre de ce travail, un petit angle de vue sera plutôt souhaité car il permettra d'avoir plus d'information concernant les véhicules. De plus, l'algorithme YOLO sera plus performant et aura moins de chance de se tromper car les cibles occuperont plus de place sur l'image.

A partir de la connaissance de l'angle de vue d'une caméra, noté β, il est possible d'estimer la distance d entre un objet et la caméra de la façon suivante. On notera que l'image est de dimensions connues $W \times H$, comme par exemple 1920×1080. De plus, il sera supposé que l'angle de vue de la caméra est horizontal et on fera donc un développement en utilisant la dimension W. Un développement analogue pourra être réalisé dans le cas d'un angle de vue vertical ou diagonal.

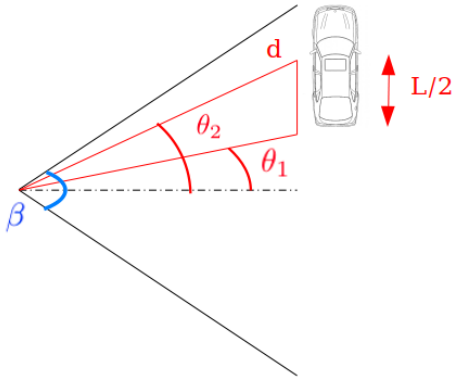


FIGURE 3.12 – Schéma illustrant la détermination de la distance caméra-véhicule

Dans le développement qui suit, la caméra est représentée par un point, tout comme la cible. Celle-ci est alors représentée par son centre de masse, déterminé de la façon expliquée à la Section 3.2.2. La distance recherchée correspond donc à la

distance entre ces deux points. Dans un premier temps, on recherche le facteur de conversion entre le monde réel et le monde des images. Ceci est donné par l'angle de vue β de la caméra. En effet, il exprime le fait que W pixels correspondent à 28°. Le facteur de conversion recherché vaut donc $\frac{\beta}{W}$. Dès lors, connaissant ce facteur de conversion, nous pouvons calculer les angles θ_1 et θ_2. Pour ce faire, on suppose que la position du centre de masse sur l'image est (x, y) pour lequel l'origine du repère est le centre de l'image. De plus, grâce à l'algorithme YOLO, nous connaissons la surface occupée par une cible sur l'image. Ceci est donné par les dimensions de la bounding box. Dans le cas d'un angle de vue horizontal, seule la dimension horizontale de la box, nommée L, nous intéressera. On a :

$$\theta_1 = \left(x - \frac{L}{2}\right)\frac{\beta}{W} \tag{3.21}$$

$$\theta_2 = x\frac{\beta}{W} \tag{3.22}$$

Finalement, connaissant la dimension horizontale réelle D du véhicule, nous pouvons obtenir l'estimation de la distance avec un peu de trigonométrie. On a :

$$d = \frac{\frac{D}{2}}{\cos(\theta_2)(\tan(\theta_2) - \tan(\theta_1))} \tag{3.23}$$

Dans l'équation ci-dessus, il est important de remarquer deux choses. La première est qu'il faut être capable de déterminer la dimension D du véhicule. Dans le cas de la simulation, celle-ci est fournie et donc, nous la connaissons avec précision. Par contre, dans le cas de la validation expérimentale, celle-ci n'est pas connue. Elle sera alors déterminée en considérant une valeur moyenne. La deuxième chose à remarquer est le fait qu'il faut être capable de déterminer l'orientation relative du véhicule par rapport à la caméra. En effet, pour un véhicule vu de face, la largeur L de la bounding box donnée par YOLO correspondra à la largeur du véhicule. Par contre, pour un véhicule vu de côté, la largeur L correspondera cette fois-ci à la longueur du véhicule. Ceci fait l'objet de la section suivante.

3.2.4 Orientation d'un véhicule par rapport à la caméra

Comme dit à la section précédente, la détermination de l'orientation relative du véhicule par rapport à la caméra est cruciale pour estimer correctement la distance caméra-objet. Concrètement, il y a trois orientations relatives possibles, ce qui est illustré à la Figure suivante :

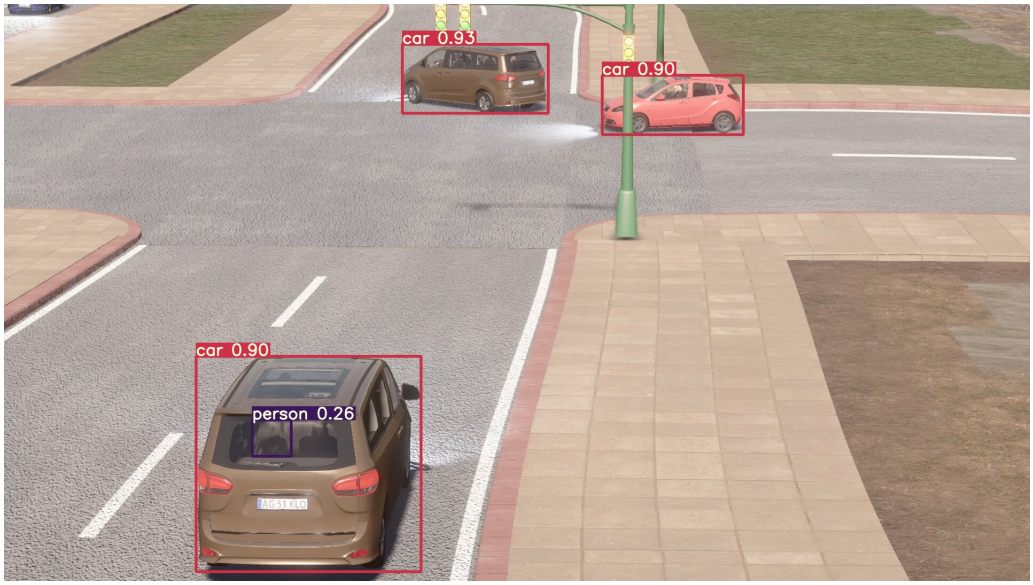


FIGURE 3.13 – Sur cette Figure, on constate que la dimension principale de la voiture rouge est sa longueur. Pour la voiture brune la plus proche de la caméra, il s'agit de sa largeur. L'autre voiture brune est quant à elle en biais et la notion de dimension principale est ambiguë.

Dans un premier temps, nous pourrions être tentés d'estimer l'orientation des véhicules sur base de l'aire de la bounding box : une "grande" aire correspondant à un véhicule dont la dimension principale est la longueur et une "petite" aire correspondant à un véhicule dont la dimension principale est la largeur. Cependant, ce genre de raisonnement ne tient pas la route. En effet, un bon nombre de véhicules seront vus en biais et donc, l'aire de la bounding box sera plus grande que nécessaire. C'est par exemple le cas sur la Figure 3.13 de la voiture brune la plus proche de la caméra : sa dimension principale est sa largeur mais la caméra voit tout de même un peu le côté du véhicule. Il en résulte donc que l'algorithme ne sera pas stable. De plus, cela demande également de définir la notion de grande et de petite aire. En reprenant la discussion précédente concernant l'angle de vue de la caméra, il en résulte que le seuil pour déterminer une "grande" aire dépend de l'angle de vue et donc, de la caméra. Cet algorithme n'est donc pas capable de généraliser les résultats. En outre, un véhicule loin de la caméra aura toujours une aire plus petite qu'un véhicule plus proche, quand bien même ils aient la même orientation.

Un algorithme plus stable et plus simple consiste à regarder le déplacement en pixel d'une voiture entre deux images. On note d_x le déplacement horizontal et d_y le déplacement vertical. Il y a trois cas de figure :

— $d_x > d_y$
Dans le cas où $d_x > d_y$, le véhicule s'est principalement déplacé horizontalement dans l'image. Par conséquent, nous aurons plutôt tendance à le voir de côté et sa dimension principale est sa longueur. C'est le cas de la voiture rouge sur la Figure 3.13.

— $d_x < d_y$
Dans le cas où $d_x < d_y$, le véhicule s'est principalement déplacé verticalement dans l'image. Par conséquent, il est vu de profil et sa dimension principale est sa largeur. C'est le cas de la voiture brune présente en bas de la Figure 3.13.

— $d_x = d_y$
Dans ce cas, soit le véhicule est à l'arrêt, soit il est en train de tourner. Il s'agit donc d'un cas particulier. Dans ce cas, il n'est pas possible de déterminer avec certitude l'orientation du véhicule sans information supplémentaire. Pour pallier ce problème, il suffit d'utiliser la distance fournie par le radar et de tester les deux dimensions principales possibles. La dimension correcte sera celle qui fournit la distance la plus proche de celle du radar. Cependant, l'algorithme peut se tromper s'il existe des erreurs en amont.

En pratique, cet algorithme, bien que simple, fournit de bons résultats dans la plupart des cas et donc, une estimation correcte de la distance objet-caméra. Bien qu'il ne traite pas toujours parfaitement les véhicules en train de tourner ou à l'arrêt, ceux-ci correspondent à des cas particuliers. On voit alors tout l'intérêt d'utiliser un système bimodal radar-caméra, suivi d'une étape de filtrage avec, par exemple, un réseau de neurones ou un filtre de Kalman. En effet, l'estimation de la distance par la caméra étant faussée dans ce cas, le filtre devra utiliser la distance fournie par le radar pour prédire la trajectoire du véhicule.

3.2.5 Optimisation de la distance objet-caméra

L'estimation de la distance entre une cible et la caméra est un point essentiel du travail. En effet, une estimation la plus précise possible est requise pour non seulement en déduire la position (x, y, z) de la cible de façon précise, mais également pour estimer le plus fidèlement possible la vitesse de ce même véhicule. Ce dernier point fera l'objet de la Section suivante. Ainsi, il est important d'améliorer cette estimation de la distance pour éviter de fausser totalement la position et la vitesse de la cible. De plus, l'association des données peut être également biaisée lorsque la distance est mal estimée. Ceci fera l'objet d'une autre section.

Concrètement, trois pistes d'amélioration de la distance ont été étudiées. Celles-ci font l'objet des sous-points suivants.

Utilisation de la hauteur du véhicule

Comme expliqué à la Section 3.2.4, déterminer l'orientation du véhicule, et donc sa dimension principale, est essentiel pour pouvoir estimer la distance caméra-objet. Cependant, pour éviter ce problème, nous pourrions utiliser la hauteur du véhicule. Néanmoins, la caméra étant en hauteur, les véhicules sont vus de haut et donc, la hauteur de la bounding box n'épouse pas correctement la hauteur du véhicule. Ceci est d'autant plus vrai pour les véhicules à proximité de la caméra, c'est-à-dire les véhicules les plus intéressants. Cet effet indésirable peut être observé à la Figure 3.13. Il a donc été décidé de ne pas utiliser cette optimisation.

Toutefois, il pourrait être intéressant de combiner cette méthode avec celle consistant à déterminer l'orientation des véhicules. En effet, dans le cas où un véhicule est en biais, l'algorithme pourrait se baser sur la hauteur du véhicule. Dans les autres cas, l'algorithme se baserait sur la dimension principale. Par manque de temps, cette méthode hybride n'a pas été utilisée. En effet, elle semble uniquement utile à la simulation. En pratique, en utilisant une caméra réelle, nous pouvons en estimer la matrice caméra. Dès lors, le problème de décision de l'orientation des véhicules ne se pose plus.

Utilisation de plusieurs points caractéristiques du véhicule

Précédemment, la méthode consistant à déterminer la distance entre une cible et la caméra se basait sur un unique point (le centre de masse du véhicule). Comme expliqué précédemment, une petite variation de position de ce point entraîne une erreur dans l'estimation de la distance. Cette petite variation pouvant être considérée comme du bruit aléatoire, l'estimation de la distance devient également aléatoire. Par le Théorème Central Limite, au plus l'algorithme utilisera des points dans l'estimation de la distance, au plus celle-ci sera correcte. En analysant l'Equation (3.23), on se rend compte que l'estimation de la distance par la caméra ne dépend que de la coordonnée x du centre de masse, exprimée dans le repère de l'image. La coordonnée y de ce point n'étant pas présente dans les équations. Par conséquent, il a été décidé d'utiliser l'ensemble des pixels de coordonnées y et nous avons pu observer une petite amélioration de l'estimation de la distance objet-caméra. Cependant, ce bénéfice ne permet que de lisser l'estimation et ne permet en aucun cas de la corriger dans le cas d'une estimation complètement fausse.

Utilisation de l'aire des bounding boxs

Un véhicule correspond à une forme 3D complexe. Une fois celle-ci capturée par une caméra, elle est représentée en 2D sur le plan de l'image. Par conséquent, l'aire d'une bouding box fournie par YOLO décrit entièrement, dans le monde des pixels, un véhicule. De plus, indirectement, elle décrit la distance et la vitesse de la cible. Sans perte de généralité, nous considérons un véhicule s'éloignant de la caméra.

Au plus un véhicule est loin de la caméra, au moins ce véhicule occupera de place sur l'image. Autrement dit, il occupera une plus petite surface. Dès lors, un véhicule s'éloignant de la caméra sera associé à une bounding box dont l'aire décroît au cours du temps. Il existe donc une fonction reliant la distance objet-caméra à la surface occupée par cet objet sur l'image. De plus, il existe également une autre fonction reliant la vitesse d'un véhicule et l'aire de la bounding box qui lui est associée. En effet, un véhicule roulant à une vitesse radiale de 10 [km/h] va s'éloigner moins vite de la caméra qu'un véhicule roulant à une vitesse radiale de 50 [km/h]. Ainsi, entre deux instants différents, le changement d'aire sera plus important pour le deuxième véhicule que pour le premier. Par conséquent, le changement d'aire d'une bounding box fournit de l'information quant à la vitesse du véhicule.

3.3 Extraction de la vitesse d'un véhicule

Comme dit dans ce qui précède, un véhicule est modélisé par son centre de masse. Sa vitesse absolue est alors donnée par :

$$v = \frac{\Delta d}{\Delta t} \tag{3.24}$$

Dans l'Equation ci-dessus, Δd correspond à la distance parcourue par un véhicule au cours d'un laps de temps Δt. Ainsi, la connaissance de la position d'un même véhicule à deux instants différents permet d'en calculer sa vitesse. En effet, un véhicule se trouvant au temps t_1 à la position (x_1, y_1, z_1) et au temps t_2 à la position (x_2, y_2, z_2) a une vitesse absolue donnée par :

$$v = \frac{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}}{t_2 - t_1} \tag{3.25}$$

Il y a deux conséquences que l'on peut retirer de cette équation. La première est que l'algorithme permettant d'estimer la vitesse est extrêmement sensible à la précision de l'estimation de la distance. En effet, une estimation erronée de la distance objet-caméra entraînera une plus grosse erreur concernant l'estimation de la vitesse. En effet, à un temps t_i, la distance objet-caméra estimée vaut :

$$\hat{d}_i = d_i + e_i \tag{3.26}$$

où e_i représente l'erreur d'estimation.

Ainsi, entre deux instants différents t_1 et t_2, le pire cas correspond au fait que la distance d_1 est sous-estimée et la distance d_2 est sur-estimée [3]. En effet, dans ce cas, les erreurs d'estimation s'additionnent et l'erreur commise devient grande.

La deuxième conséquence de l'Equation (3.25) est qu'au plus la différence de temps $\Delta t = t_2 - t_1$ est petite, au plus l'estimation de la vitesse absolue d'un véhicule va correspondre à l'estimation de la vitesse instantanée de ce même véhicule. Le problème est que, comme dit précédemment, l'estimation de la vitesse est fortement dépendante de l'estimation de la distance. En reprenant les notations de l'Equation (3.26), l'estimation de la vitesse est donnée par :

$$v = \frac{\Delta d + e_1 + e_2}{t_2 - t_1} \tag{3.27}$$

En prenant deux instants t_1 et t_2 proches, les erreurs d'estimation e_1 et e_2 ne sont pas négligeables car la distance parcourue Δd est faible. Par contre, si ces instants sont un peu plus éloignés, alors les erreurs d'estimation deviennent négligeables face à la distance parcourue. Cependant au plus ces instants seront éloignés, au plus il sera compliqué d'associer un même véhicule sur deux images différentes et donc, au plus l'algorithme de tracking risque de se tromper. On constate donc qu'il y a un compromis à réaliser entre l'erreur provenant de l'estimation de la distance et l'erreur provenant de l'algorithme de tracking. Dans notre cas, le système est bimodal et nous avons accès à un radar dont une des fonctions est d'estimer la vitesse d'un des véhicules. Etant donné que, par la suite, nous filtrerons les données avec un filtre de Kalman, un Multi-Layer Perceptron (MLP) ou un Transformer, il est plus judicieux de ne pas tolérer les erreurs provenant de l'algorithme de tracking. Ainsi, dans le cas d'une estimation erronée de la vitesse, il faudra filtrer un peu plus les données provenant de la caméra, c'est-à-dire apporter davantage de confiance aux données provenant du radar. Par ailleurs, il est important de noter que les performances liées aux filtrages dépendent tout de même de la précision de l'estimation de la distance. En effet, dans le cas idéal où il n'y a pas d'erreur liée ni à l'estimation de la distance, ni d'erreur provenant de l'algorithme de tracking, le filtrage sera parfait et nous aurons une estimation de la vitesse parfaite.

3. S'il existe une erreur dans les coordonnées sphériques représentant la position d'un véhicule, cette même erreur se retrouvera lors du passage vers les coordonnées cartésiennes.

3.4 Méthode d'association des cibles

L'association des données radar-caméra est un point important et suppose une synchronisation parfaite entre ces deux senseurs. Pour ce faire, un algorithme robuste a été implémenté. A un instant donné, il considère une donnée d'un senseur et la compare par rapport aux données de l'autre senseur au même instant. Il va alors associer les deux senseurs en utilisant la méthode des moindres carrés. Ensuite, l'algorithme supprime cette donnée pour n'avoir qu'une seule association. Une amélioration de cet algorithme consiste à implémenter un tracker beaucoup plus performant tel SORT. L'inconvénient de ceci est que l'extraction des données caméra demandera beaucoup plus de temps.

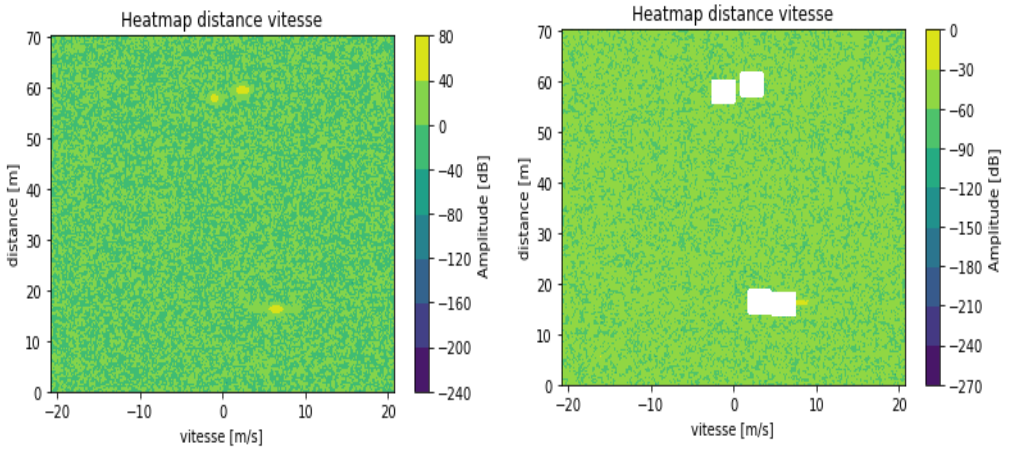
3.5 Détermination du maximum local dans la heatmap distance-vitesse

Si un véhicule se trouve dans le champ de vison d'un radar, celui-ci apparaît en tant que maximum local sur une carte thermique distance-vitesse. Si N véhicules sont présents, il y a donc N maxima locaux. Dès lors, pour identifier le nombre de véhicules présents sur une carte thermique, il faut en déterminer le nombre de maxima locaux.

Un algorithme simple à implémenter mais suffisamment robuste consiste à

1. normaliser les valeurs telles que $X_{norm} = \frac{X-\mu}{\sigma}$ où μ est la moyenne et σ l'écart-type des amplitudes trouvées ;
2. rechercher l'amplitude maximale ;
3. calculer la moyenne et la variance de la zone aux alentours du point maximal trouvé
4. Si cette moyenne et cette variance sont plus grandes qu'un seuil défini, alors l'indice de ligne et de colonne de cette valeur maximale est converti en vitesse-distance puisqu'il existe une relation distance-ligne et vitesse-colonne. Sinon, passer à l'étape suivante ;
5. mettre la zone alentour à zéro afin de ne pas identifier deux fois un même véhicule ;
6. itérer jusqu'à ce que la valeur maximale soit inférieure à un seuil défini.

L'étape 3 sert à vérifier que la valeur maximale se trouve dans un amas de points et ainsi correspond bien à un véhicule. Notons que les valeurs-seuils ont été trouvées par l'analyse de données réelles.



(a) Carte thermique distance-vitesse

(b) Carte thermique après le passage de l'algorithme de recherche

FIGURE 3.14

3.6 Détermination du maximum local dans la heatmap des angles

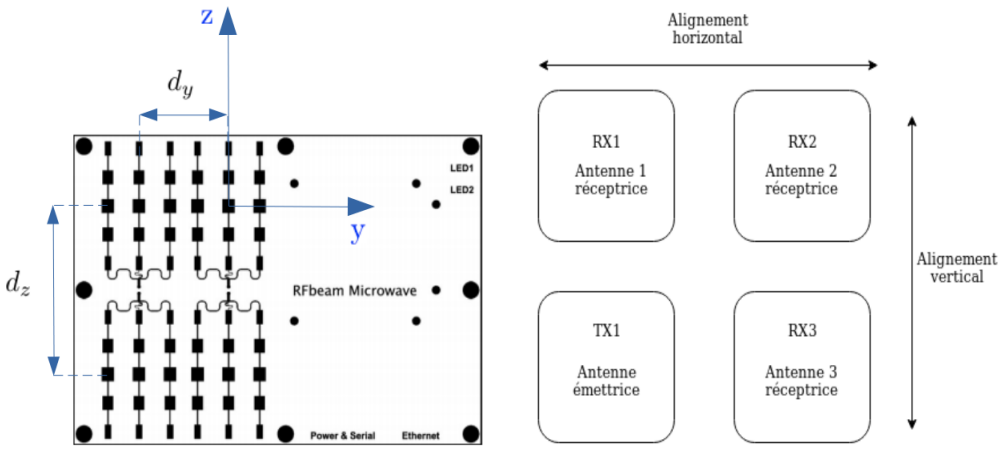
Une fois les maxima locaux identifiés, on connaît leurs indices [ligne, colonne] que l'on renommera [N,M]. Ils correspondent au N$^{\text{ième}}$ échantillon du M$^{\text{ième}}$ chirp reçu. Grâce à eux, chaque véhicule détecté pourra être associé à une carte thermique des angles θ, φ.

Comme expliqué dans la section 1.1.4, il existe un délai de réception entre les ondes provenant d'une direction particulière. Dans notre cas, le radar est composé de 3 antennes dont les espacement sont représentés par la Figure 3.15a.

Pour retrouver cette direction de propagation, il faut compenser ce délai entre les signaux reçus par les 3 antennes et maximiser cette somme. Donc, il faut faire une recherche exhaustive de tous les angles (u, v) de telle sorte à maximiser l'équation suivante :

$$S(u,v) = argmax_{\hat{u}} \sum_{i=1}^{3} A_i e^{-jkd_i\hat{u}} \tag{3.28}$$

Cependant, à cause des ambiguïtés, on limite la recherche à l'intervalle [0,u_{ambi}] et [0,v_{ambi}]. En trouvant la valeur maximale dans cet intervalle à l'aide des indices de ligne et colonne, il nous est possible d'identifier les angles φ et θ en appliquant



(a) Définition de l'espacement des angles avec d_y = 21.8 [mm] et d_z = 39.6 [mm] [19]

(b) Alignement physique des antennes dans le plan OYZ [7]

FIGURE 3.15 – Espacement des antennes du radar

les opérations (3.30) aux valeurs de u et v trouvées.

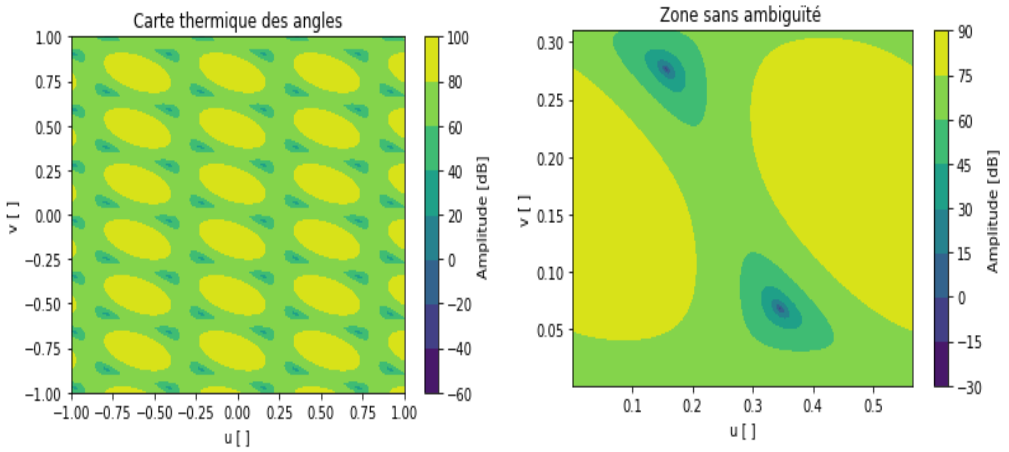
$$\theta = arccos(v) \tag{3.29}$$

$$\varphi = arctan\left(\frac{u}{\sqrt{1-u^2-v^2}}\right) \tag{3.30}$$

Comme nous pouvons le constater sur la Figure 1.11, les ambiguïtés sont périodiques aussi bien selon l'angle θ que selon l'angle φ.

3.6.1 Levée de l'ambiguïté

On remarque que les angles θ et ϕ seront bornés entre respectivement [0,v_{ambi}] et [0,u_{ambi}] (Figure 3.16b). Pour lever ces ambiguïtés, on utilise les données fournies par la caméra. Après avoir associé chaque détection caméra à une détection radar, grâce à une association par la distance minimale, on sait lever l'ambiguïté. En effet, comme les ambiguïtés sont périodiques d'intervalle $\frac{d_i}{\lambda}$, on rajoute un facteur n entier à cet intervalle afin d'obtenir la bonne valeur. Tout cela en supposant que l'association soit correctement faite.



(a) Maximisation de la corrélation

(b) Limitation de la zone de recherche à la zone sans ambiguïté

FIGURE 3.16

3.7 Conclusion data

Ainsi, à la fin de ce processus, nous obtenons un vecteur 8D (d,v,θ, ϕ) représentant la position dans l'espace ainsi que la vitesse radiale de chacune des cibles détectées. Celui-ci étant bruité à cause du hardware, la localisation des véhicules n'est pas toujours précise. Une meilleure précision peut-être obtenue en appliquant un filtre de Kalman ou un réseau de neurones de type Multi-Layer Perceptron (MLP) ou Transformer.

Pour conclure ce chapitre, nous considérons l'image illustrée à la Figure suivante et nous en extrayons les données.

FIGURE 3.17 – Exemple d'image capturée par une caméra

Sur cette image, seul le véhicule le plus proche est considéré, les autres se situant trop loin du système radar-caméra. Nous obtenons les données suivantes :

Données de la caméra				Données du radar			
X [m]	Y [m]	Z [m]	v [m/s]	X [m]	Y [m]	Z [m]	v [m/s]
19.6506	51.6341	-6.9446	0.9873	22.8509	59.1195	-8.58525	-1.3020

TABLE 3.1 – Résultats obtenus pour chaque senseur pour le véhicule le plus proche illustré à la Figure 3.17

X [m]	Y [m]	Z [m]	v [m/s]
21.1525	55.2186	-8.0539	3.8335

TABLE 3.2 – Position réelle du véhicule

Comme nous pouvons le constater, les positions estimées par le radar et la caméra sont fiables. L'application d'un filtre sur ces données permettra une meilleure prédiction de la trajectoire des véhicules. Par contre, on constate que la caméra a plus de mal pour estimer la vitesse (radiale) du véhicule.

Chapitre 4

Filtrage des données

Ce chapitre se concentre sur le filtrage des données fournies par les différentes modalités. Pour ce faire, trois architectures sont utilisées : un filtre de Kalman, un perceptron multi-couches (MLP) et un Transformer. La comparaison de ces trois architectures permet d'atteindre simultanément deux objectifs. D'une part, cela permet de déterminer si la non-linéarité des architectures de type MLP et Transformer fournit de meilleurs résultats qu'une architecture linéaire de type filtre de Kalman. D'un autre côté, cela permet de déterminer si une architecture plus conséquente, telle que celle d'un Transformer, fournit de meilleurs résultats qu'une architecture plus basique, telle que celle d'un MLP. Ces architectures seront optimisées en fonction de l'erreur quadratique moyenne. Par la suite, on suppose qu'il n'y a pas de souci de synchronisation et que les données extraites des deux senseurs sont homogènes.

4.1 Filtre de Kalman

Un filtre de Kalman est un modèle linéaire qui possède plusieurs avantages. Citons par exemples sa facilité d'implémentation ainsi que sa faible consommation en ressources calculatoires, ce qui le rend plus apte à une application en temps réel. D'un autre côté, l'un de ses désavantages est une limitation des paramètres utilisés en raison du modèle linéaire choisi.

L'implémentation du filtre de Kalman ainsi que l'étape d'association se basent sur [7]. Les différentes étapes de fonctionnement du filtre ont cependant été modifiées et quelques hyperparamètres ont été changés. Une technique de fusion des données des deux senseurs a également été ajoutée.

4.1.1 Définition du modèle

Le vecteur d'état utilisé pour décrire l'état des objets à un instant t est donné par :

$$\vec{x}_t = \begin{bmatrix} d & \theta & \varphi & \dot{d} & \dot{\theta} & \dot{\varphi} \end{bmatrix}^T \tag{4.1}$$

où

d [m]	la distance radiale
θ [°]	l'angle d'élévation
φ [°]	l'angle azimutal
$\dot{d}$ [m/s]	la vitesse radiale
$\dot{\theta}$ [°/s]	la vitesse angulaire d'élévation
$\dot{\varphi}$ [°/s]	la vitesse angulaire azimutale

TABLE 4.1 – Signification des variables du vecteur d'état

Les paramètres ci-dessus caractérisent entièrement la position et la vitesse d'un objet dans l'espace à un instant t donné. Afin de suivre le déplacement des objets à chaque pas de temps, des pistes sont utilisées. Celles-ci sont des repères permettant d'assigner une identité à chaque détection reçue et d'en identifier des nouvelles.

Le filtre de Kalman peut traiter plusieurs détections en même temps : chaque détection est associée à une piste i qui correspond à une cible. Si aucune piste n'est présente à cette étape, les étapes d'association et d'actualisation ne peuvent avoir lieu et on passe donc directement à l'étape de filtrage qui gère la création et l'initialisation des pistes.

4.1.2 Fonctionnement du filtre de Kalman

Un filtre de Kalman comporte quatre étapes :

1. Prédiction : le modèle linéaire est utilisé pour prédire l'emplacement à l'instant t de chaque piste et la matrice de covariance est actualisée.
2. Association : les détections sont associées aux pistes existantes.
3. Actualisation : la matrice de covariance est actualisée grâce aux détections reçues.
4. Filtrage : de nouvelles pistes sont crées pour les détections associées à aucune piste et les pistes ayant atteint l'âge maximal sont supprimées.

Les hyperparamètres présents dans le filtre de Kalman sont :

L'âge maximal de la piste	I_{max}
L'indice de confiance d'un véhicule	P_D
Le paramètre de pondération de la probabilité de faux positif	λ
L'indice de distance maximale entre une piste et une détection	β
Le critère de distance probabiliste minimale	d_{min}

TABLE 4.2 – Hyperparamètres du filtre de Kalman

Parmi ces hyperparamètres, certains seront plus analysés dans le Section 4.1.4. Ceux non abordés auront la valeur par défaut attribuée par [7]. Par la suite, on définit $\vec{x}$, $\tilde{x}$ et $\hat{x}$ respectivement comme le vecteur d'état connu, prédit et estimé.

Etape de prédiction

L'étape de prédiction consiste à prédire, à partir du vecteur d'état $\vec{x}_{t-1}$, le vecteur d'état à l'instant t. Pour ce faire, on utilise la matrice $\mathbf{F}$ reliant l'état $t-1$ à l'état t. L'état prédit s'exprime alors comme :

$$\tilde{x}_t = \mathbf{F}\hat{x}_{t-1} + \vec{q} \quad \text{avec} \quad \mathbf{F} = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

où le vecteur $\vec{q}$ correspond à l'erreur associée au modèle à chaque itération.

Une matrice de covariance $\mathbf{P}$ est associée à chaque vecteur d'état. Celle-ci exprimant l'incertitude globale du vecteur d'état à l'instant t. A chaque itération, cette matrice $\mathbf{P}$ varie et est mise à jour à l'aide de la règle suivante :

$$\mathbf{\tilde{P}_{t|t-1}} = \mathbf{F_t}\mathbf{\hat{P}_{t|t-1}}\mathbf{F^T} + \mathbf{Q} \tag{4.3}$$

avec $\mathbf{Q}$ la matrice de covariance du bruit du processus, modélisé par du bruit blanc gaussien.

Étape d'association

Dans cette étape, on associe les détections j aux pistes i existantes à l'aide d'un critère de distance minimale β. La première étape dans le calcul de cette distance

consiste à estimer l'erreur $v_t^{i,j}$ entre chaque piste $\tilde{x}_{i,t}$ et chaque mesure $\vec{z}_{j,t}$:

$$v_t^{i,j} = \vec{z}_{j,t} - \mathbf{H_t}\tilde{x}_{i,t} \tag{4.4}$$

Le vecteur de mesures $\vec{z}_{j,t}$ étant un vecteur de dimensions 8 fournissant la distance, les deux angles ainsi que la vitesse radiale pour les deux senseurs, la technique de fusion sera expliquée à la Section 4.1.3. La matrice $\mathbf{H}_t$ est la matrice d'observation permettant de définir les mesures observées par les senseurs. Ceux-ci étant homogènes, elle est définie par :

$$\mathbf{H_t} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.5}$$

Ensuite la matrice $\mathbf{S_{i,t}}$ est créée. Elle permet l'estimation de l'erreur totale grâce à l'addition de l'incertitude sur la positon de la piste avec l'incertitude de la mesure $\mathbf{R}$.

$$\mathbf{S_{i,t}} = \mathbf{H_t}\tilde{\mathbf{P}}_{\mathbf{i,t}}\mathbf{H_t^T} + \mathbf{R_t} \tag{4.6}$$

La distance probabiliste $\hat{d}_{i,j}$ se calcule alors comme

$$\left(\hat{d}_{i,j}\right)^2 = \left(v_t^{i,j}\right)^T \mathbf{S_{i,t}}^{-1} v_t^{i,j} \tag{4.7}$$

A l'aide de cette équation, on observe que plus l'erreur entre la mesure et la prédiction est grande, plus la distance probabiliste sera petite. Ainsi une distance $\hat{d}_{i,j}$ est calculée entre chaque piste existante i et chaque nouvelle détection j. Ces dernières sont assignées à la piste dont la distance est minimale et inférieure à un paramètre β. Plusieurs détections peuvent dès lors être associées à une même piste.

Étape d'actualisation

L'étape d'actualisation consiste à adapter la valeur prédite en se basant sur la mesure. On commence par calculer la probabilité pondérée de chaque détection j associée à une piste i en considérant qu'il y a m détections associées à une même piste i :

$$p^{i,j} = \frac{e^{-0.5(\hat{d}^{i,j})^2}}{\lambda\frac{1-P_D}{P_D}\sqrt{|\mathbf{S_{i,t}}|} + \sum_{j=0}^{m} e^{-0.5(\hat{d}^{i,j})^2}};\ \ p^{i,j} \leq 1 \text{ et } \sum_i p^{i,j} = 1 \tag{4.8}$$

Dans cette équation, le terme $\lambda\frac{1-P_D}{P_D}$ sert à prendre en compte les fausses détections positives où les variables P_D et λ sont des hyperparamètres permettant

de régler le seuil de faux positifs. L'Equation (4.8) se comporte comme une fonction softmax. Plus la distance probabiliste $d_{i,j}$ est grande, plus la probabilité que l'association entre la détection j et la piste i est correcte. La probabilité $p^{i,0}$ que les mesures soient des faux-positifs s'exprime alors comme :

$$p^{i,j} = \frac{\lambda \frac{1-P_D}{P_D} \sqrt{|\mathbf{S_{i,t}}|}}{\lambda \frac{1-P_D}{P_D} \sqrt{|\mathbf{S_{i,t}}|} + \sum_{j=0}^{m} e^{-0.5(\hat{d}^{i,j})^2}};\ p^{i,j} \tag{4.9}$$

Il faut noter que nous ne prenons pas en compte l'indice de confiance des classes de véhicules. Nous supposons dès lors l'indice P_d comme constant et égal à 0.95.

Par ailleurs, le gain de Kalman $\mathbf{K}_i$ permet d'établir le poids entre la prédiction $\tilde{x}_{i,t}$ et les mesures $\vec{z}_{j,t}$. Il est défini par :

$$\mathbf{K_i} = \tilde{\mathbf{P}}_{\mathbf{i,t}} \mathbf{H_t} \mathbf{S}_{\mathbf{i,t}}^{-\mathbf{1}} \tag{4.10}$$

Le gain de Kalman peut donc être interprété comme l'importance que l'on donne à la mesure par rapport à la prédiction. S'il se rapproche de la matrice identité, alors les détections sont considérées comme plus importante que les valeurs prédites, tandis que s'il se rapproche de la matrice nulle, alors les mesures ne sont pas considérées pour l'actualisation du vecteur d'état estimé. La position de la piste i estimée peut donc s'écrire comme :

$$\hat{x}_{i,t} = \tilde{x}_{i,t} + K_i \sum_{j=1}^{m} p^{i,j} \tilde{v}_t^{i,j} \tag{4.11}$$

Finalement, la matrice de covariance est actualisée grâce à l'équation suivante :

$$\hat{\mathbf{P}}_{\mathbf{i,t}} = p^{i,0} \tilde{\mathbf{P}}_{i,t} \tag{4.12}$$

$$+ (1 - p^{i,0}) \left(\tilde{\mathbf{P}}_{i,t} - \mathbf{K_i} \mathbf{S_{i,t}} \mathbf{K_i^T} \right) \tag{4.13}$$

$$+ \mathbf{K}_i \left(\sum_{j=1}^{m} v_t^{i,j} p^{i,j} (v_t^{i,j})^T - (1 - p^{i,0}) \left(\sum_{j=1}^{m} v_t^{i,j} p^{i,j} \right) \left(\sum_{j=1}^{m} v_t^{i,j} p^{i,j} \right)^T \right) \mathbf{K}_i^T \tag{4.14}$$

Cette équation est trouvée en utilisant l'équation d'actualisation $\mathbf{P_{k|k}} = (\mathbf{I} - \mathbf{K_k H_k}) \mathbf{P_{k|k-1}}$ du modèle théorique simple du filtre de Kalman et en pondérant les termes avec la probabilité associée à chaque piste-détection i,j. Le premier terme est la matrice de covariance estimée, pondérée par la probabilité de faux positif. Le deuxième terme indique que la matrice de covariance actualisée est proportionnelle au gain de Kalman si des mesures sont associées à cette piste. Le troisième terme montre que plus la variance entre les détections est grande, plus la matrice de covariance actualisée augmentera.

Etape de filtrage

Dans cette étape, les détections non associées sont transformées en nouvelles pistes et les pistes n'étant plus associées à aucune nouvelle détection depuis I_{max} itérations sont supprimées.

4.1.3 Fusion des données

Les données captées par le radar et la caméra sont extraites et jointes dans le vecteur de mesures $z_{j,t}$. Celui-ci est un vecteur de 8 dimensions comportant la distance radiale, les angles d'élévation et d'azimut ainsi que la vitesse radiale fournis par les deux senseurs (la caméra et le radar). Dans les autres architectures, les données captées par le radar et la caméra sont fusionnées à l'aide d'un réseau de neurone, donc d'une boite noire traitant les données grâce à des poids trouvés lors de l'entraînement du réseau. Dans le filtre de Kalman, le modèle utilisé assume l'hypothèse que le vecteur de mesures $z_{j,t}$ est composé de quatre éléments observés. Il faut donc fusionner les données captées par les deux senseurs qui sont homogènes, c'est-à-dire qui mesurent les mêmes éléments mais avec un dégrée d'erreur diffèrent. Cette incertitude sur l'erreur est représentée par une matrice de covariance de mesures $\mathbf{R}$ pour chaque senseur.

La stratégie de fusion des mesures proposée se base sur [20]. Dans cet article, deux méthodes de fusion sont comparées. Seule la deuxième méthode proposée sera utilisée. Celle-ci consiste à fusionner les données à l'aide d'une pondération se basant sur la minimisation de l'erreur quadratique de la matrice R de covariance de chaque senseur de telle manière que :

$$\mathbf{z_t} = \left[\sum_{j=1}^{N} R_j^{-1}\right]^{-1} \sum_{j=1}^{N} R_j^{-1} z_{j,t} \tag{4.15}$$

$$\mathbf{H_t} = \left[\sum_{j=1}^{N} R_j^{-1}\right]^{-1} \sum_{j=1}^{N} R_j^{-1} H_t \tag{4.16}$$

$$\mathbf{R_t} = \left[\sum_{j=1}^{N} R_j^{-1}\right]^{-1} \tag{4.17}$$

Cette méthode n'est valable que lorsque les matrices d'observation des deux senseurs sont homogènes. En outre, on note R_c et R_r les matrices de covariance de

la caméra et du radar. Les équation deviennent donc :

$$\mathbf{z_t} = \left[R_r^{-1} + R_c^{-1}\right]^{-1} \left(R_r^{-1} z_{t,r} + R_c^{-1} z_{t,c}\right) \tag{4.18}$$

$$\mathbf{H_t} = \left[R_r^{-1} + R_c^{-1}\right]^{-1} \left(R_r^{-1} H_{t,r} + R_c^{-1} H_{t,c}\right) \tag{4.19}$$

$$\mathbf{R_t} = \left[R_r^{-1} + R_c^{-1}\right]^{-1} \tag{4.20}$$

Ainsi, ces équations permettent d'obtenir le vecteur de mesures $z_{j,t}$, la matrice d'observation H_t, ainsi que les matrices de covariance totale de l'erreur pour les deux senseurs.

4.1.4 Choix des paramètres

Cette section consiste à la détermination des paramètres optimisant l'erreur quadratique moyenne (MSE) du filtre de Kalman. En effet, les paramètres utilisés dans [7] correspondent à des paramètres empiriques tandis que nos simulations sont générées synthétiquement.

Matrice de covariance du bruit de mesure des senseurs

En pratique, il faudrait déterminer les matrices de covariance des senseurs en utilisant différentes techniques, comme expliqué dans [21]. Or, dans notre cas, les données étant simulées, l'accès à leur position à un instant t est connu. En prenant comme hypothèse que ces matrices sont diagonales et que le bruit est du bruit blanc gaussien, il suffit de calculer la variance de la différence entre les données mesurées par les senseurs et les données réelles. On a :

$$\mathbf{R_r} = \begin{bmatrix} 11 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0128 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.353 & 0 & 0 & 0 \\ 0 & 0 & 0 & 13.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.21}$$

$$\mathbf{R_c} = \begin{bmatrix} 49 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0075 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.0025 & 0 & 0 & 0 \\ 0 & 0 & 0 & 136 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.22}$$

En pratique, étant donné que ces matrices doivent être inversibles, on remplace les éléments nuls dans la diagonale par un facteur $\epsilon = 10^{-10}$.

Matrice de covariance du bruit du processus

La matrice Q représente l'incertitude de l'erreur du modèle choisi. Le modèle fonctionnant sur un pas de temps de $\frac{1}{30}$ [s], la distance de déplacement entre deux instants est faible. En se basant sur [16], qui lui utilisait un pas de temps de 1 centième de seconde, les valeurs de $\mathbf{Q}$ sont adaptées en multipliant les valeurs par $\frac{10}{3}$ afin d'obtenir une variance équivalente.

$$\mathbf{Q} = \begin{bmatrix} 0.167 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0033 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.0033 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.0167 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.167 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.167 \end{bmatrix} \tag{4.23}$$

Le paramètre P_D

Cet indice est utilisé pour l'alarme de faux négatifs. Etant donné que l'indice de confiance fourni par *YOLO* n'est pas utilisé, nous avons décidé de choisir un indice de confiance fixe et valant 0.95. Pour choisir cette valeur, nous avons changé itérativement sa valeur sur un même jeu de données en calculant la MSE des trajectoires prédites par les différentes pistes.

En observant la Figure 4.1, on remarque que ce facteur n'a pas une grande influence sur la MSE, excepté lorsque $P_D = 1$. Dans le cadre des valeurs générées synthétiquement, les données analysées ne contiennent pas de faux positif. Néanmoins ces derniers risquent d'apparaître lors de la validation expérimentale. Dès lors, nous avons gardé la valeur de P_D à 0.95 au lieu de 1 pour prendre ce cas en compte.

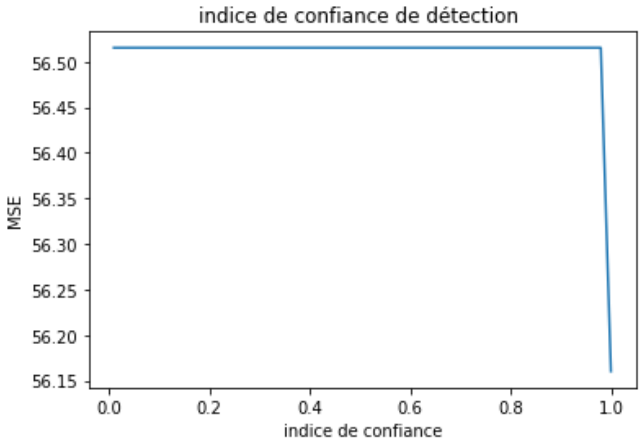


FIGURE 4.1 – MSE par indice de confiance

Age maximal des pistes

L'âge maximal des pistes intervient dans l'étape de filtrage et permet de sélectionner et supprimer les pistes associées à aucune détection depuis I_{max} itérations. La Figure ci-dessous illustre l'impact de ce paramètre :

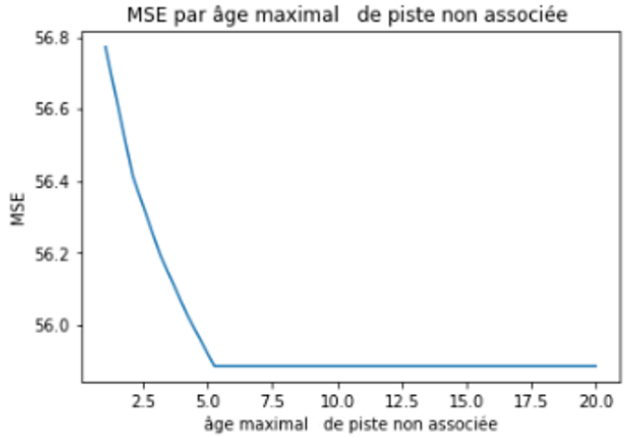


FIGURE 4.2 – MSE par âge maximal de piste non associée

Cette figure a été créée en calculant la MSE et en faisant varier le paramètre I_{max}. La Figure ci-dessus montre que la MSE diminue jusqu'à stagner vers un âge maximal de 5. Ceci indique que, si une piste reste non associée à une détection pendant plus de 5 itérations, elle sera effacée.

Le critère de distance probabiliste

Le critère de distance probabiliste intervient dans l'étape de filtrage. Il permet de limiter le nombre de nouvelles pistes créées. En effet, il faut que la distance probabiliste calculée soit supérieure au *threshold* afin qu'une nouvelle piste soit créée. Cela réduit la probabilité que des pistes de trop faible distance probabiliste soient assignées en tant que nouvelles pistes.

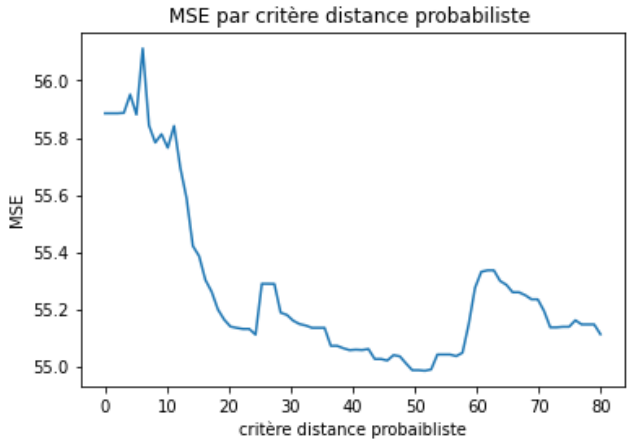


FIGURE 4.3 – MSE en fonction du *threshold*

4.1.5 Résultats

Une fois les paramètres du filtre de Kalman optimisés, nous pouvons filtrer les données fournies par le radar et la caméra et estimer des trajectoires. La Figure suivante compare la trajectoire estimée de trois véhicules (en bleu) par rapport à leur trajectoire réelle (en rouge).

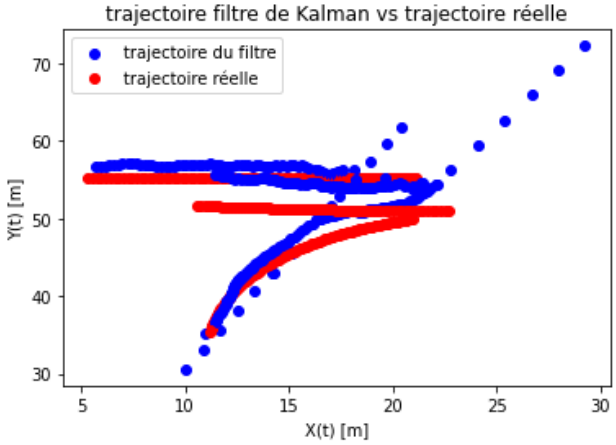


FIGURE 4.4 – Exemple de trajectoires estimées à l'aide d'un filtre de Kalman. On observe trois trajectoires distinctes correspondant à trois véhicules différents.

Sur la Figure ci-dessus, on constate que le filtre de Kalman arrive à distinguer correctement les trois cibles différentes. Les trajectoires estimées correspondent approximativement aux trajectoires réelles. On constate cependant que le filtre de Kalman a plus de difficultés à évaluer la trajectoire de cibles en train de tourner. On observe aussi des points partant dans un sens précis. Ils correspondent à des pistes n'ayant aucune détection associée.

4.2 Multi-Layer Perceptron

Le perceptron multi-couches, Multi-Layer Perceptron (MLP) en anglais, est un réseau de neurones inventé en 1957 par Frank Rosenblatt. Aujourd'hui, ce type de réseau est largement utilisé, tant pour faire de la régression non-linéaire qu'en tant que sous-module d'un autre réseau de neurones plus grand et plus complexe, comme par exemple le Transformer. Il est également à la base du machine learning moderne, à l'instar de la rétropropagation du gradient dont nous parlerons par la suite. Traditionnellement, un perceptron multi-couches est fait d'une couche d'entrée, d'une couche cachée de neurones et finalement, d'une couche de sortie. Sur la Figure suivante, un MLP est schématisé. Le schéma provient de [22].

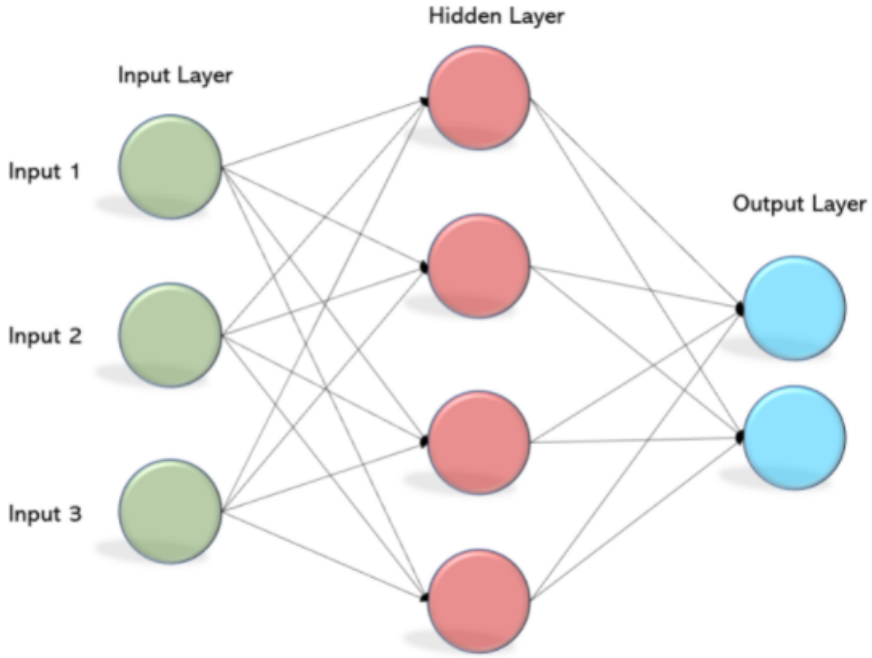


FIGURE 4.5 – Le perceptron multi-couches est un réseau de neurones qui a en général deux couches. Sur cette Figure, la couche d'entrée correspond à la couche où l'on vient connecter les valeurs d'entrée. De même, la couche de sortie correspond à la couche donnant les sorties de la régression effectuée par le réseau. La couche cachée est faite de neurones qui vont permettre de faire avancer l'information de l'entrée vers la sortie

Comme nous pouvons le constater sur la Figure ci-dessus, ce type de réseau peut fournir plusieurs sorties. Ceci est particulièrement utile dans le cadre de ce travail étant donné que nous souhaitons prédire quatre sorties : la position (x, y, z) d'une

cible ainsi que sa vitesse. Par ailleurs, nous pouvons remarquer qu'une particularité de ce réseau est que l'information est traitée de la couche d'entrée vers la couche de sortie. On parle de réseau à propagation directe (Feed Forward Network). De plus, il est bon de constater que le nombre de neurones des deux couches internes peut être différent. Enfin, une caractéristique notable de ce réseau est que la sortie d'un neurone correspond à une des entrées de l'ensemble des neurones de la couche suivante. On parle de Fully Connected Network.

Concrètement, la prédiction d'une ou de plusieurs sorties fonctionne de la façon suivante. A la couche d'entrée, on connecte plusieurs entrées qui correspondent aux données sur lesquelles le perceptron multi-couches va devoir se baser. Chacune de ces données va alors être une entrée des neurones constituant la couche interne. A ce niveau, l'information a donc atteint un premier niveau d'abstraction. Ensuite, le même schéma se reproduit : la sortie de chaque neurone de la couche constitue une entrée de chaque neurone constituant la deuxième couche. La sortie de cette couche correspond aux prédictions. Ainsi, l'information présente en entrée du réseau est traitée par tous les neurones. Cependant, les connexions entrée-sortie ne sont pas toute traitées de la même façon : un poids matérialise la pondération de la connexion. Ainsi, lors de la résolution d'un problème, il faudra déterminer les poids optimaux de chacune des liaisons inter-neuronales. Ceci est réalisé grâce à la rétropropagation du gradient qui sera expliquée plus tard.

4.2.1 Modélisation d'un neurone

Un réseau de neurones sert à modéliser le cerveau humain lors de la réalisation d'une tâche bien précise, comme de la classification ou de la régression. Ainsi, un neurone artificiel modélise le fonctionnement d'un neurone humain. Ceci est représenté à la Figure suivante provenant de [23] :

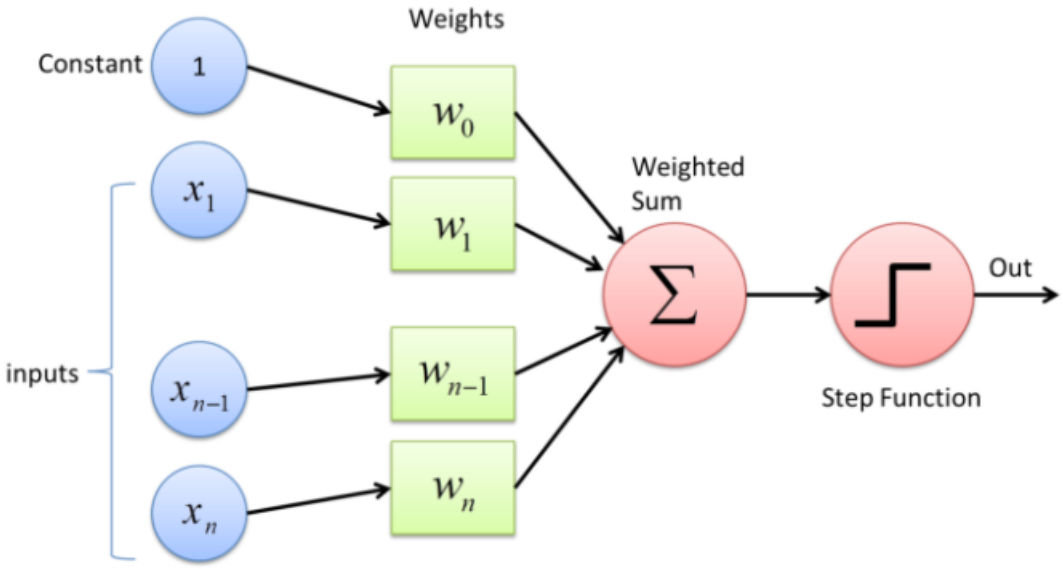


FIGURE 4.6 – Le fonctionnement d'un neurone artificiel se fait en deux étapes : d'abord il combine l'ensemble des entrées x_i, ensuite cette combinaison passe à travers une fonction d'activation ψ.

Dans cette section, nous supposons que les poids W_{ij} sont connus. Dès lors, un neurone combine l'ensemble des entrées x_i de la façon suivante :

$$a_j = \psi\left(\sum_{i=1}^{n} x_i\right) \tag{4.24}$$

où l'indice j correspond à la couche j du réseau.

On constate donc que la sortie d'un neurone est une fonction de la combinaison linéaire de toutes les entrées de ce même neurone. Ce modèle est alors utilisé pour tous les neurones constituant le réseau. Ainsi, un élément essentiel du réseau est la fonction d'activation ψ utilisée par chacun des neurones. C'est cet élément qui permet d'apporter le comportement non-linéaire d'un réseau de neurones. Aujourd'hui, trois fonctions d'activation sont largement utilisées, la fonction ReLU étant la plus utilisée :

1. La fonction sigmoïde définie par :

$$\psi(x) = \frac{1}{1+e^{-x}} \tag{4.25}$$

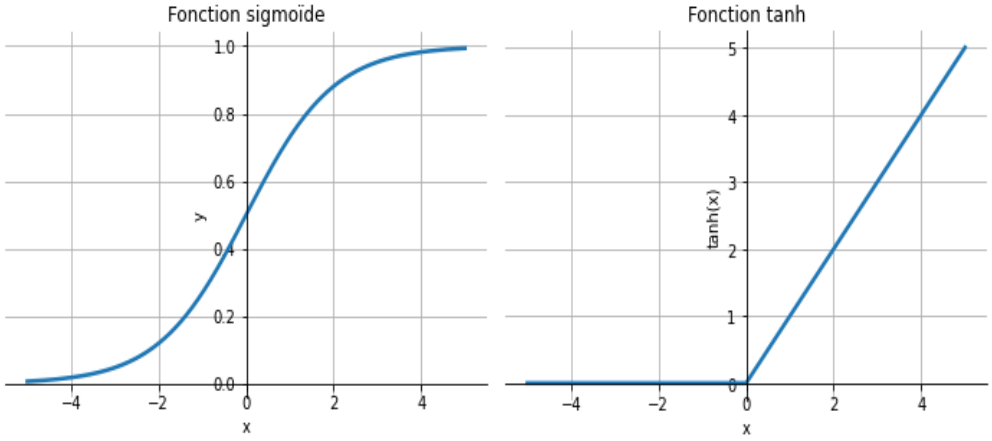
2. La fonction tangente hyperbolique définie par :

$$\psi(x) = \tanh x = \frac{2}{1+e^{-2x}} - 1 \tag{4.26}$$

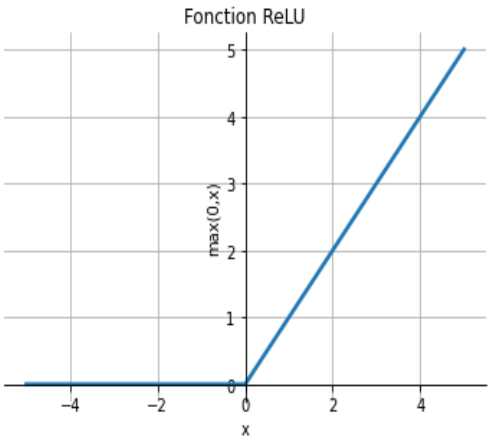
3. La fonction Unité de Rectification Linéaire ReLU (Rectified Linear Unit) définie par :

$$\psi(x) = \begin{cases} 0 \text{ si } x < 0 \\ x \text{ si } x \geq 0 \end{cases} \tag{4.27}$$

En plus d'être toutes les trois non-linéaires, ces fonctions partagent trois autres propriétés communes : elles sont monotones, valent 0 à l'origine et sont partout différentiables. Cette dernière propriété est cruciale pour la rétropropagation du gradient. Les Figures suivantes illustrent ces fonctions.



(a) Fonction sigmoïde (b) Fonction tangente hyperbolique



(c) Fonction ReLU

FIGURE 4.7 – Cette Figure illustre les trois fonctions d'activation les plus couramment utilisées.

Comme nous pouvons le constater sur la Figure 4.7, seules les fonctions sigmoïde et tangente hyperbolique réalisent un seuillage de l'entrée. Par ailleurs, le choix de la fonction d'activation dépend de la tâche que doit réaliser le réseau de neurones. Etant donné que la fonction ReLU est la plus utilisée, il a été décidé de l'utiliser par la suite.

4.2.2 Rétropropagation de l'erreur

Comme vu dans la section précédente, les connexions inter-neuronales sont établies avec des poids. Ces poids sont déterminés de façon optimale lors de la phase d'entraînement du réseau. Dans cette section, nous nous concentrerons sur le critère d'erreur et l'expression de son gradient. Dans la Section 4.2.3, nous nous concentrerons sur les méthodes possibles pour optimiser les poids, connaissant le gradient du critère d'erreur à chaque couche. Ce critère d'erreur à optimiser est défini de la façon suivante :

$$E = (y - \hat{y})^2 \tag{4.28}$$

où $\hat{y}$ correspond aux valeurs prédites par le modèle.

La phase d'entraînement du réseau se fait par rétropropagation de l'erreur définie par l'Equation (4.28). Afin d'expliquer cette phase, nous utilisons les notations telles qu'exprimées à la Figure suivante :

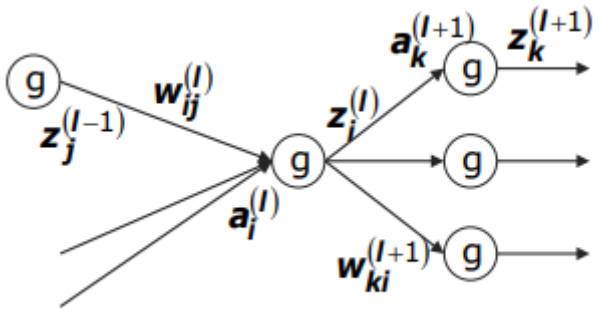


FIGURE 4.8 – Cette Figure représente deux neurones et provient de [24]. La sortie du neurone j de la couche $l-1$ est notée z_j^{l-1} et l'entrée du neurone i de la couche suivante l est notée a_i^l.

Au vu des notations exprimées par la Figure 4.8, on peut exprimer l'entrée du neurone i de la couche l par une combinaison linéaire des sorties des neurones de la couche précédente. Pour simplifier les notations et sans perte de généralité, on

supposera que les couches intermédiaires ont toutes n neurones. Le raisonnement suivant provient de [24]. On a :

$$a_i^l = \sum_{j=1}^{n} W_{ij}^l z_j^{l-1} \tag{4.29}$$

Dans un premier temps, on cherche à déterminer l'expression du gradient $\frac{\partial E}{\partial W_{ij}^l}$. Par la règle de la chaîne, on a :

$$\frac{\partial E}{\partial W_{ij}^l} = \frac{\partial E}{\partial a_i^l}\frac{\partial a_i^l}{\partial W_{ij}^l} \tag{4.30}$$

$$= \frac{\partial E}{\partial a_i^l} z_j^{l-1} \tag{4.31}$$

On définit alors pour le neurone i de la couche l : $\frac{\partial E}{\partial a_i^l} \triangleq \delta_i^l$. Pour les neurones de la dernière couche, ce terme est connu. En effet, on a :

$$\delta_i^l = \frac{\partial E}{\partial a_i^l} \tag{4.32}$$

$$= \frac{\partial E}{\partial y_i^l}\frac{\partial y_i^l}{\partial a_i^l} \tag{4.33}$$

Dans l'équation ci-dessus, le terme y_i^l correspond à la sortie i fournie par la neurone i de la couche l. Par conséquent, en considérant le critère d'erreur (4.28) et le fait que le neurone a une fonction d'activation ψ, on a :

$$\delta_i^l = -2(t_i - y_i^l)\psi'(a_i^l) \tag{4.34}$$

Ici, la fonction ψ' correspond à la dérivée de la fonction d'activation d'un neurone. De plus, le terme t_i correspond à la valeur qui devrait être produite à la sortie i dans le cas idéal. On voit donc que pour la dernière couche du réseau, le gradient $\frac{\partial E}{\partial W_{ij}^l} = \delta_i^l z_j^{l-1}$ est entièrement connu. Pour déterminer l'expression du gradient dans les couches intermédiaires, on va retropropager le gradient. Autrement dit, connaissant le gradient de la couche $l+1$, on va calculer le gradient de la couche l. En reprenant les notations de la Figure 4.8, comme le perceptron multi-couches est un réseau complètement connecté, on sait que la sortie du neurone i de la couche l sera l'entrée de l'ensemble des neurones de la couche $l+1$. On a donc :

$$\delta_i^l = \frac{\partial E}{\partial a_i^l} \tag{4.35}$$

$$= \sum_{k=1}^{n} \frac{\partial E}{\partial a_k^{l+1}}\frac{\partial a_k^{l+1}}{\partial a_i^l} \tag{4.36}$$

Connaissant le terme a_k^{l+1}, on montre facilement que l'on a :

$$\delta_i^l = \sum_{k=1}^{n} \delta_k^{l+1} W_{ki}^{l+1} \psi'(a_i^l) \tag{4.37}$$

Dans cette équation, on constate donc que le terme δ_i^l de la couche l est déterminé grâce à la connaissance des termes de la couche suivante $l+1$. Par conséquent, lors de la phase d'entraînement du réseau, la rétropropagation du terme d'erreur E permet de déterminer le gradient à chacune des couches du réseau de neurones. Par ailleurs, il est bon de noter que cette méthode n'est pas spécifique au MLP. En effet, dans bien d'autres réseaux, ce principe est utilisé pour permettre de calculer les poids des connexions inter-neuronales. De plus, dans cette section, le critère d'erreur tel que défini par l'Equation (4.28) est arbitraire. Ce principe de rétropropagation des poids fonctionne très bien sur d'autres critères tel que l'entropie croisée. Néanmoins, dans le cadre de ce travail, le MLP sert à réaliser une régression non-linéaire et non une classification. Par conséquent, ce dernier critère n'est pas adéquat pour ce travail. Finalement, connaissant l'expression du gradient, plusieurs méthodes d'optimisation peuvent être utilisées de façon à optimiser les poids. Ceci fera l'objet de la section suivante.

4.2.3 Méthodes d'entraînement des réseaux de neurones

Bien que de nombreuses méthodes d'optimisation existent, nous nous concentrerons sur trois méthodes en particulier. Nous les testerons ensuite, tant dans le cas du perceptron multi-couches que dans le cas du Transformer. Dans un premier temps, nous expliquerons la descente de gradient et la descente de gradient stochastique et dans un second temps, nous discuterons de la méthode d'Adam.

Descente de gradient

Lors d'une descente de gradient, les poids sont mis à jour à chaque itération selon la règle suivante :

$$W(t+1) = W(t) - \alpha \left. \frac{\partial E}{\partial W} \right|_{W(t)} \tag{4.38}$$

Dans l'expression ci-dessus, le paramètre α correspond au taux d'apprentissage. La détermination de son expression est cruciale : s'il est trop grand, l'algorithme risque de ne pas converger. S'il est trop petit, l'algorithme risque de demander énormément de temps pour converger. La détermination de ce paramètre sera discutée dans la Section 4.2.4. De plus, ce paramètre peut être soit constant, soit variable au cours du temps. Un défaut majeur de cette méthode est qu'elle demande

l'utilisation de toutes les données pour faire une seule itération. Par conséquent, utiliser un grand nombre de données demandera beaucoup de temps. Autrement dit, cette méthode est plutôt requise pour les petits ensembles de données.

Descente de gradient stochastique

La descente de gradient stochastique applique la même règle que celle de la descente de gradient. Cependant, la différence est qu'une itération ne considère qu'une seule donnée et non pas l'ensemble des données disponibles. Autrement dit, à chaque itération, la méthode applique la règle donnée par l'Equation (4.38) sur une donnée et, ensuite, met à jour les poids. Par conséquent, on atteint une convergence plus rapidement avec cette méthode. Cependant, les résultats après convergence sont plus entachés de bruit. Par ailleurs, pour que cette méthode fonctionne correctement, il faut mélanger les données de départ. Ainsi, l'algorithme choisira aléatoirement une donnée et on ne risque pas d'avoir un certain ordre prédéfini dans les données.

Descente de gradient avec des mini-batches

Un batch est un sous-ensemble de l'ensemble des données de départ. Ainsi, la descente de gradient avec mini batches consiste à appliquer la règle de la descente de gradient sur un sous-ensemble de données (un batch) et non pas sur toutes les données ou sur seulement une seule donnée. Ceci permet donc d'avoir un certain compromis entre la descente de gradient et la descente de gradient stochastique. En effet, cette méthode aura une convergence plus rapide que la descente de gradient et aura moins de bruit après convergence que la descente de gradient stochastique. C'est cette méthode qui sera utilisée lorsque nous appliquerons une descente de gradient pour le perceptron multi-couches ou pour le Transformer.

Méthode d'Adam

La méthode d'Adam est une extension de la descente stochastique de gradient. Contrairement à cette dernière méthode qui applique un même taux d'apprentissage α pour tous les poids, cette nouvelle méthode applique un taux d'apprentissage différent pour chacun des poids. Concrètement, elle calcule des moyennes mobiles exponentielles du gradient et du gradient au carré. Ceci est donné par la règle suivante pour un poids w :

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{4.39}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{4.40}$$

Dans ces deux équations ci-dessus, m_t et v_t sont une estimation biaisée des moments d'ordre 1 et d'ordre 2. Le terme g_t, quant à lui, représente le gradient à l'itération t pour un certain poids. La suite de l'algorithme consiste à calculer les moyennes et mettre à jour les paramètres. On a :

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{4.41}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{4.42}$$

$$w^t = w_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{4.43}$$

Dans les équations ci-dessus, les termes β_1^t et β_2^t correspondent à la t-ème puissance de β_1 et β_2. Par ailleurs, le paramètre ϵ sert à éviter une division par zéro et est en général très petit. En général, on prendra les valeurs suivantes : $\epsilon = 10^{-8}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$. Par la suite, nous considérerons ces valeurs comme acquises et nous ne les optimiserons pas.

Comme nous pouvons le constater sur la Figure suivante, la méthode d'Adam fonctionne mieux sur un grand ensemble de données lorsque l'on cherche à entraîner un perceptron multi-couches.

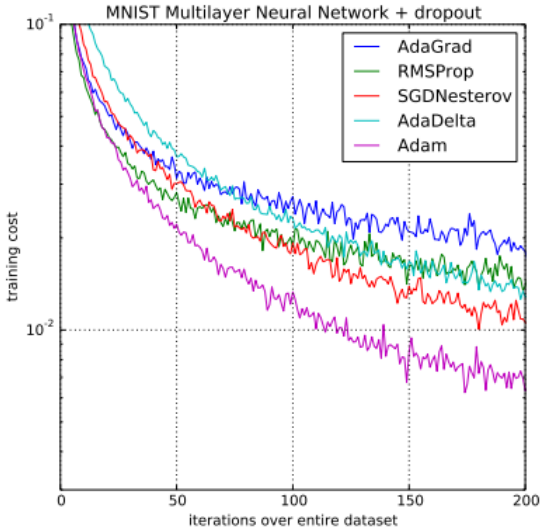


FIGURE 4.9 – Comme nous pouvons le constater sur cette Figure, la méthode d'Adam (violet) obtient de meilleurs résultats que les autres méthodes. Ces résultats proviennent de [25].

4.2.4 Paramètres choisis pour le MLP

Un perceptron multi-couches contient de nombreux paramètres. Nous n'allons cependant tester l'influence que d'un seul de ces paramètres : le nombre de neurones pour la couche intermédiaire. Ensuite, pour le cas optimal, nous regarderons l'impact d'autres paramètres : la méthode d'optimisation (descente de gradient stochastique ou Adam), le taux d'apprentissage et la taille des batchs. Nous optimiserons le MLP vis-à-vis de l'erreur quadratique moyenne (MSE).

Par ailleurs, il est bon de rappeler une notion essentielle de machine learning. Lorsque l'on souhaite entraîner et tester un réseau de neurones, on utilise un *training set* reprenant les données utilisées lors de l'entraînement ainsi qu'un *test set* correspondant aux données utilisées pour tester les performances du réseau. Ces deux ensembles de données sont, bien évidemment, différents en général, au plus il y a de données dans le training set, au mieux seront les performances. Cependant, lorsque l'on entraîne un réseau de neurones avec beaucoup de données, un effet indésirable peut se produire : le sur-entraînement (overfitting). Ceci signifie que le réseau prédit très bien lors de la phase d'entraînement mais très mal lors de la phase de test sur de nouvelles données. Autrement dit, il n'a rien appris. Ainsi, les performances d'un réseau tels que le perceptron multi-couches et le Transformer seront toujours évaluées sur un test set différent de l'ensemble des données utilisées lors de l'entraînement.

Dans un premier temps, nous considérons un perceptron multi-couches traditionnel avec une couche intermédiaire de neurones. Nous souhaitons déterminer le nombre optimal de neurones composant cette couche. Pour ce faire, nous allons tester le réseau avec un nombre différent de neurones composant la couche intermédiaire. Le cas optimal sera celui pour lequel la fonction coût lors de la phase de test sera minimale. Ceci est illustré à la Figure suivante :

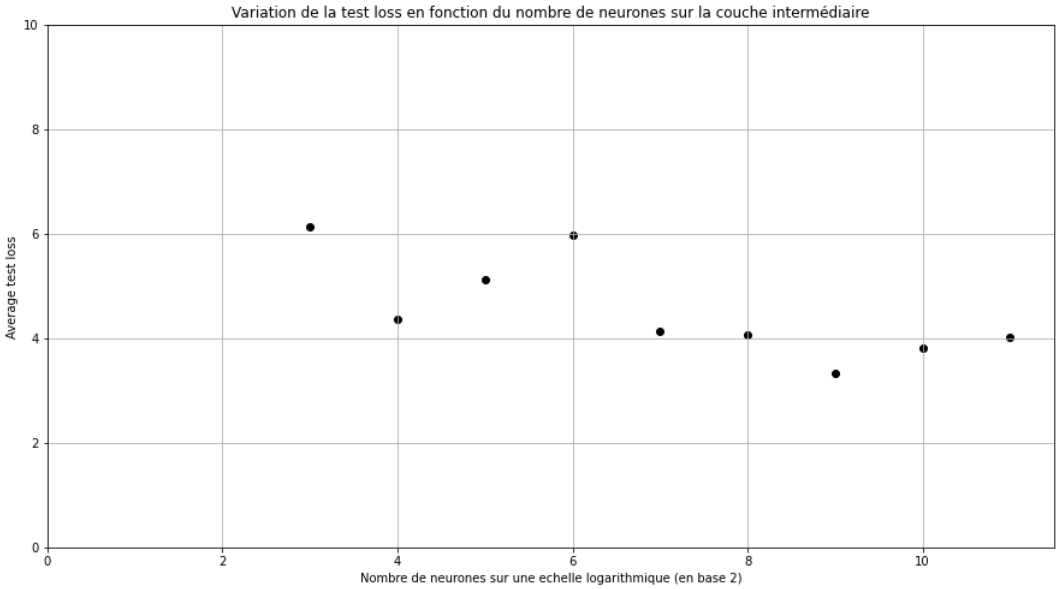


FIGURE 4.10 – Cette Figure représente les performances du réseau en fonction du nombre de neurones présents dans sa couche intermédiaire. On constate que le cas optimal correspond à $N = 512$.

Sur la Figure ci-dessus, on constate qu'il faut un nombre tout de même élevé ($N = 512$) de neurones pour obtenir les meilleurs résultats. Comme nous le verrons à la Section 4.2.5, il est possible de réduire ce nombre, et donc le temps de calcul, en fournissant un peu plus d'information au réseau. Ceci peut s'expliquer par le fait que les performances ne varient pas énormément entre un nombre de neurones $N = 2^7 = 128$ et $N = 2^{11} = 2048$. Ceci peut exprimer un léger sur-entraînement du réseau dû au fait qu'il a des données de faibles dimensions en entrée.

4.2.5 Influence du passé

Dans cette section, nous allons regarder si un perceptron multi-couches fournit de meilleurs résultats lorsqu'on lui donne non pas une seule donnée mais plusieurs instants du passé également. Pour ce faire, pour chaque véhicule, nous donnons au MLP la valeur détectée par la caméra à l'instant t_0 et les valeurs du passé correspondant aux instants t_{-1} à t_{-8}. Etant donné que la caméra travaille à une vitesse de 30 images par seconde, l'intervalle de temps entre deux images vaut $\Delta t = \frac{1}{30} = 33.3$ [ms]. Donner 8 instants du passé revient donc à fournir en entrée du MLP une séquence de 0.266 [s]. Pour un réseau à une couche intermédiaire, le nombre optimal de neurones peut être visualisé sur le graphe suivant :

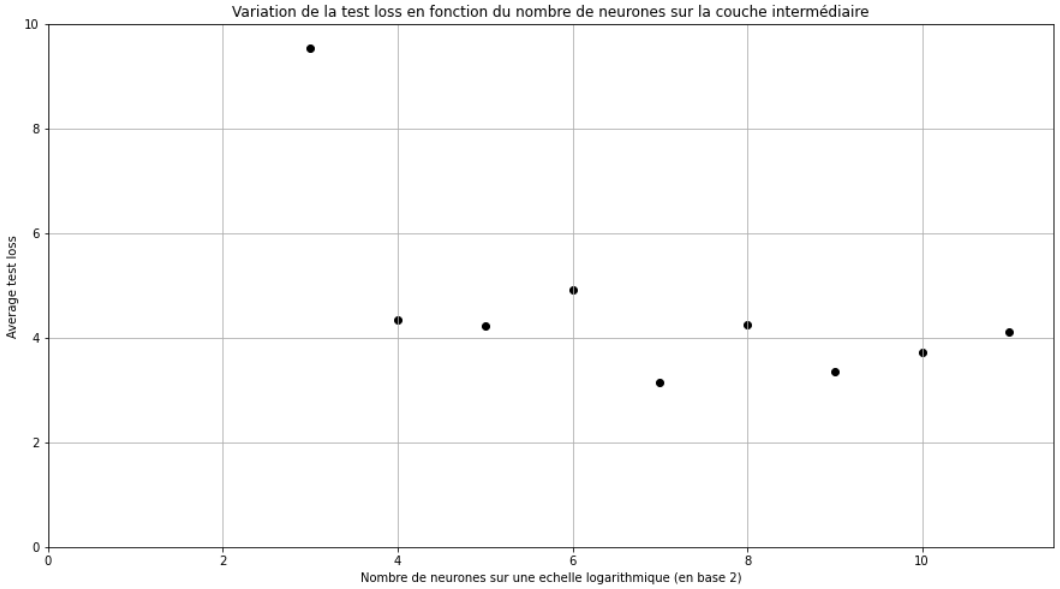


FIGURE 4.11 – Sur cette Figure, on teste un nombre différent de neurones pour la couche intermédiaire et ce, lorsque l'on fournit en entrée des valeurs du passé. On peut constater que $N = 128$ correspond au nombre de neurones optimal.

Sur la Figure ci-dessus, on constate que la fonction coût lors du test est minimale lorsque le nombre de neurones de la couche cachée du MLP vaut $N = 128$. On peut donc constater qu'un très grand nombre de neurones, comme $N = 2048$ ne permet pas de fournir de meilleures performances. Ceci peut se voir comme du sur-entraînement. En effet, s'il y a énormément de neurones, il y a énormément de poids à estimer. Par conséquent, ces nombreux poids vont être plutôt optimaux pour les données d'entraînement mais, lors de la phase de test, ils ne sauront plus suffisamment généraliser. On constate que le sur-entraînement apparaît à partir de $2^9 = 512$ neurones sur la couche intermédiaire.

Par ailleurs, on remarque que pour le cas $N = 8$, la fonction coût est maximale. Autrement dit, il s'agit du pire cas où le réseau prédit le moins bien. Toutefois, on pouvait s'y attendre. En effet, il s'agit d'un cas de sous-entraînement du réseau (underfitting). Ceci correspond simplement au fait que le réseau ne possède pas assez de paramètres que pour pouvoir généraliser suffisamment bien. Finalement, la Figure 4.11 montre des résultats auxquels on pouvait s'attendre. La fonction coût lors de la phase de test est grande lorsqu'il y a trop peu ou trop de neurones.

Comme dit précédemment, les résultats présents sur la Figure 4.11 ont été obtenus en fournissant au MLP une séquence temporelle de 0.266 [s]. Cependant, une séquence temporelle plus longue, à l'instar de 3 [s], améliore un peu les résultats.

Ceci est illustré à la Figure suivante, où nous donnons toujours 8 valeurs du passé en entrée :

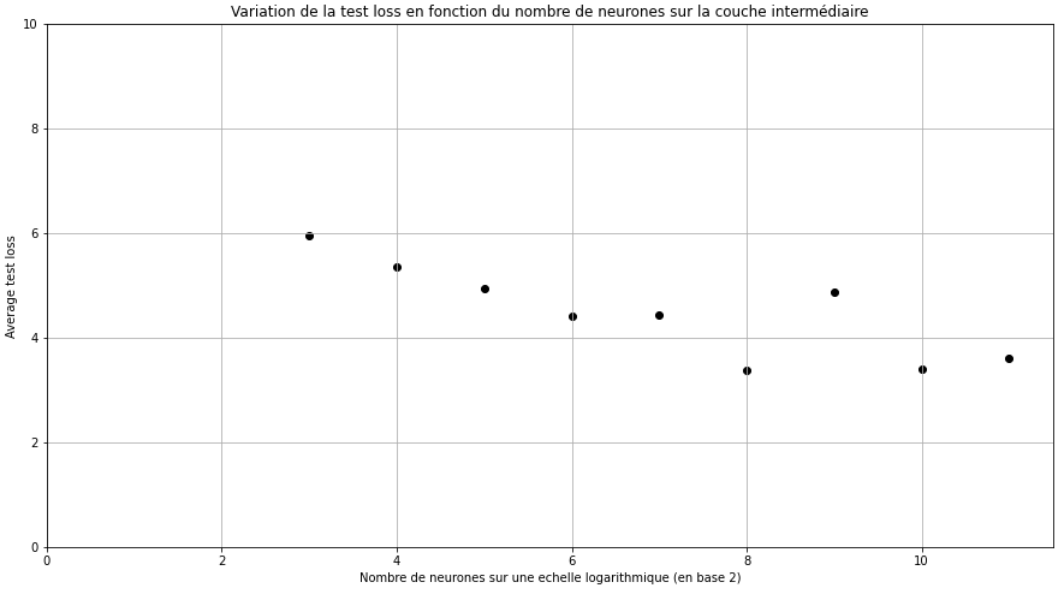
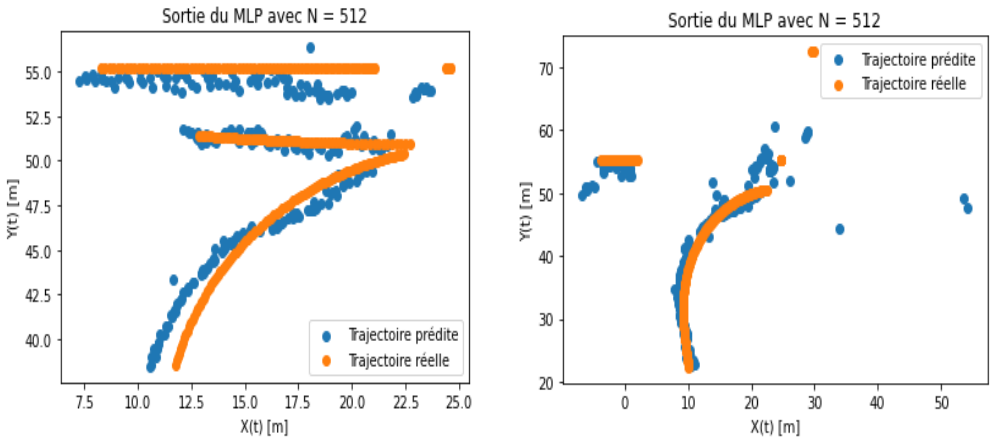


FIGURE 4.12 – Sur cette Figure, on illustre les performances du MLP lorsque l'on fournit une séquence temporelle plus longue. On constate que, globalement, les résultats sont un peu meilleurs que ceux de la Figure 4.11. Cette fois-ci, le cas $N = 128$ n'est plus le cas optimal mais fournit tout de même de bons résultats.

Ainsi, on constate donc que fournir une séquence temporelle plus longue au MLP améliore un peu les résultats. Intuitivement, sur une séquence temporelle plus longue, on connaît mieux la cinématique de la cible. On peut donc supposer que le MLP parvient à mieux prédire car il apprend un peu mieux la cinématique du véhicule. De façon raisonnable, on peut également supposer que fournir plus d'instants du passé ou une séquence temporelle encore plus longue permettra d'améliorer encore un peu plus les résultats. Néanmoins, il ne faut pas fournir une trop longue séquence pour qu'au final, le réseau apprenne très bien la cinématique des véhicules lors de la phase d'entraînement. En effet, dans ce cas-là, il risque d'entrer en sur-entraînement et donc, il ne sera plus capable de généraliser du tout ! Par ailleurs, nous pouvons également constater que le cas de sous-entraînement observé à la Figure 4.11 est réduit.

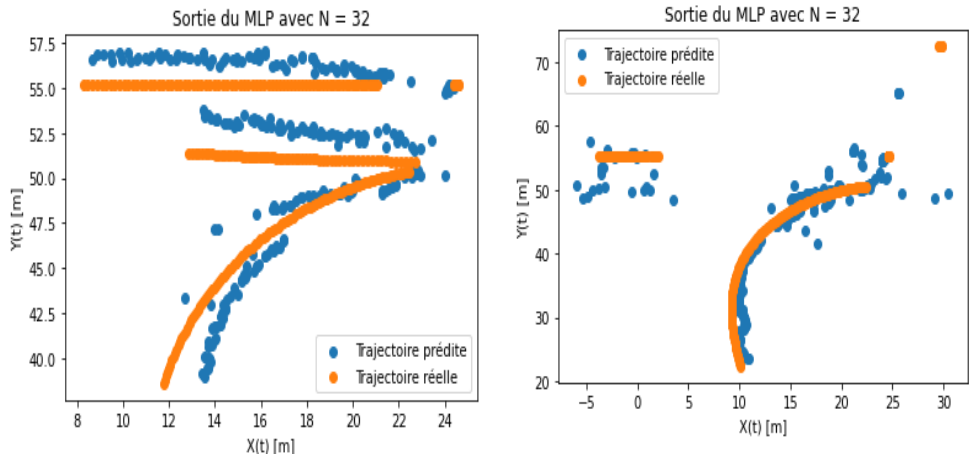
Finalement, on constate que fournir une séquence temporelle au réseau permet globalement de réduire le temps de calcul du réseau. En effet, le graphe à la Figure 4.12 (où l'on a fourni des échantillons du passé) présente des résultats plus constants que ceux de la Figure 4.10 (où il n'y a pas d'influence du passé). Ainsi, on peut

se permettre d'utiliser un réseau plus petit (au sens du nombre de neurones de la couche intermédiaire) et tout de même avoir de bons résultats. Ceci est illustré sur les Figures ci-dessous où l'on montre des trajectoires prédites par le réseau (en bleu) et les trajectoires réelles (en orange).



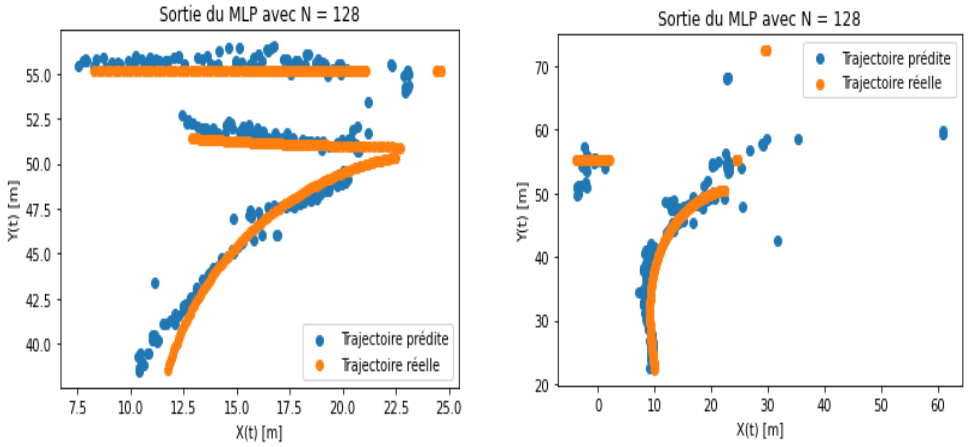
(a) (b)

FIGURE 4.13 – Ces deux Figures illustrent des prédictions de trajectoire dans le cas optimal ($N = 512$) et lorsqu'on n'utilise pas d'échantillon du passé.



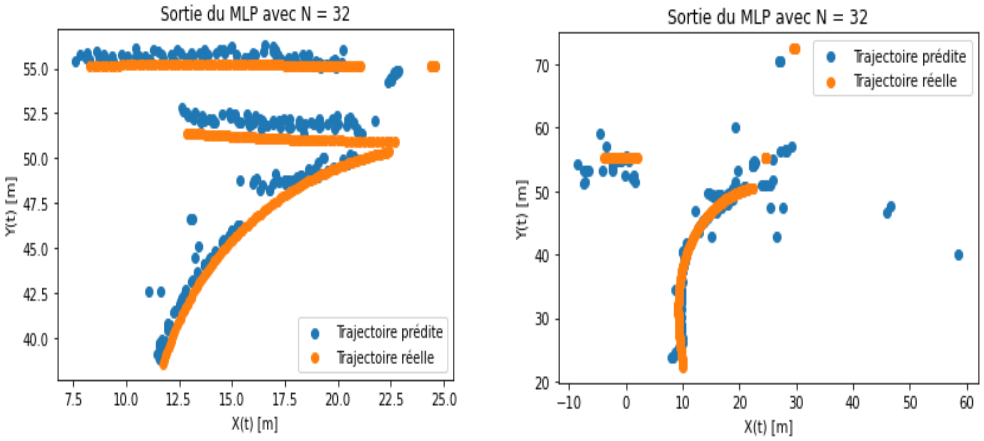
(a) (b)

FIGURE 4.14 – Ces deux Figures illustrent des prédictions de trajectoire dans un cas non-idéal ($N = 32$) et lorsqu'on n'utilise pas d'échantillon du passé.



(a) (b)

FIGURE 4.15 – Ces deux Figures illustrent des prédictions de trajectoire dans le cas idéal ($N = 128$) et lorsqu'on utilise des échantillons du passé.



(a) (b)

FIGURE 4.16 – Ces deux Figures illustrent des prédictions de trajectoire dans un cas non-idéal ($N = 32$) et lorsqu'on utilise des échantillons du passé.

Comme nous pouvons le constater sur la Figure 4.14, varier le nombre de neurones change fortement les performances du réseau lorsqu'on ne lui fournit aucun échantillon du passé. En effet, les prédictions des trajectoires sont plutôt différentes de celles fournies dans le cas idéal, tel qu'illustré sur la Figure 4.13. Par contre, si nous faisons le même test lorsqu'on fournit des échantillons du passé, on constate que les performances du réseau ne sont pas beaucoup impactées. Ceci est

illustré aux Figures 4.15 et 4.16. Par conséquent, fournir une séquence temporelle au perceptron multi-couches permet de rendre beaucoup plus robuste le réseau. Ceci peut s'expliquer par le fait que le réseau reçoit, en entrée, une plus grande dimension.

4.2.6 Influence de la méthode d'optimisation et du taux d'apprentissage

Les résultats montrés sur les Figures 4.13 à 4.16 ont tous été obtenus avec un taux d'apprentissage $\alpha = 10^{-3}$ ainsi que la méthode d'Adam. Comme nous pouvons le constater sur la Figure 4.17, la fonction coût décroît de façon monotone, ce qui peut être interprété comme un bon choix de taux d'apprentissage. Dans toute cette section, nous utiliserons un MLP à une couche intermédiaire sur laquelle il y a 128 neurones. De plus, nous fournirons une séquence temporelle de 3 [s], étant donné que ceci permet de rendre plus robuste le réseau.

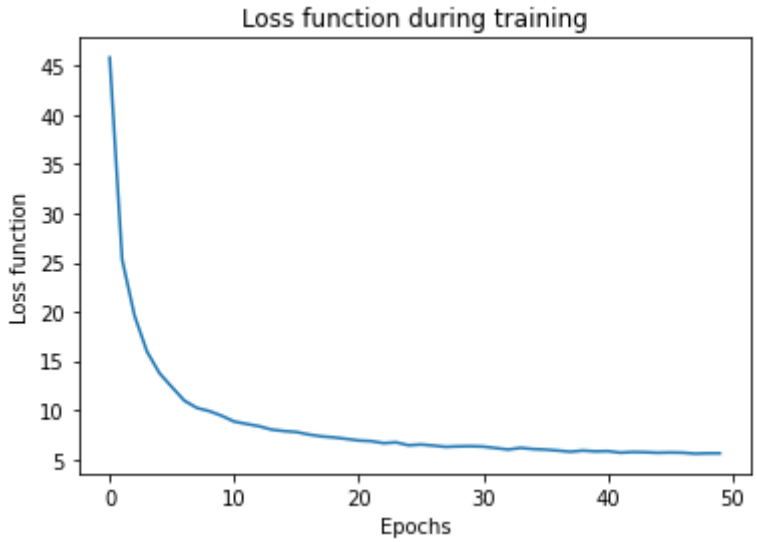


FIGURE 4.17 – Cette fonction coût a été obtenue pour un taux d'apprentissage $\alpha = 10^{-3}$. On constate qu'elle décroît de façon monotone.

Comme nous pouvons le constater, la fonction coût décroît de façon monotone. Ceci résulte d'un bon choix du taux d'apprentissage. On constate qu'il faut environ 50 *epochs*, c'est-à-dire passer 50 fois sur toutes les données, pour que le modèle converge.

Le taux d'apprentissage est un hyper-paramètre très important. Il dicte de combien vont être mis à jour les poids à chaque itération. S'il a une petite valeur,

on va peu changer les poids mais on aura besoin de plus d'epochs, et donc de temps, pour pouvoir converger. On risque ainsi de ne jamais converger. A l'inverse, si ce paramètre est grand, le modèle va converger plus rapidement mais risque de converger vers un minimum local et non pas vers le minimum global ! Par conséquent, il faut donc bien estimer la valeur de ce paramètre pour que les résultats soient aux rendez-vous. Sur la Figure suivante, différentes valeurs de taux d'apprentissage sont montrées.

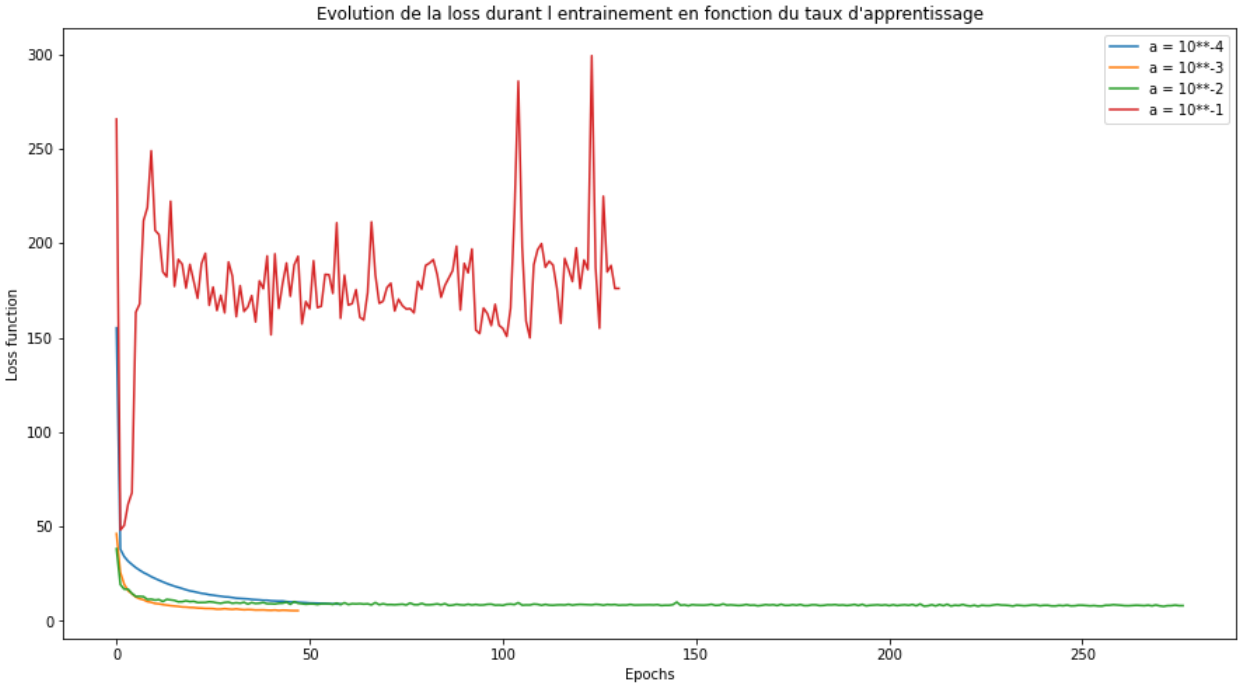


FIGURE 4.18 – Sur cette Figure, plusieurs valeurs de taux d'apprentissage sont testées. On constate qu'une valeur trop grande (en rouge) implique que la fonction coût ne converge pas vers le bon optimum. A l'inverse, une valeur trop petite requiert davantage d'itérations pour converger. L'ensemble de ces résultats ont été obtenus avec la méthode d'Adam.

Comme nous pouvons le constater sur la Figure ci-dessus, la fonction coût varie énormément en fonction du taux d'apprentissage. Lorsqu'il est trop grand ($\alpha = 10^{-1}$), le modèle ne converge pas vers l'optimum global mais plutôt vers un optimum local. Il en résulte qu'il ne va pas prédire correctement. A l'inverse, un taux d'apprentissage trop petit ($\alpha = 10^{-4}$) demande beaucoup plus d'itérations pour converger et donc, le modèle demande plus de temps pour fournir une prédiction correcte. Ainsi, sur base de la Figure 4.18, nous pouvons conclure qu'un taux d'apprentissage $\alpha = 10^{-3}$ semble un bon compromis car il fournit à la fois de bonnes

performances et ne demande pas trop de temps lors de la phase d'entraînement du réseau. Par ailleurs, la courbe en rouge exprime généralement le fait que le taux d'apprentissage est trop élevé. En effet, il s'agit d'un signal en dents de scie tel que la fonction coût augmente et diminue incessamment.

Sur la Figure suivante, nous analysons l'impact de la méthode d'optimisation. Pour ce faire, nous entraînons un perceptron multi-couches (qui a 128 neurones sur sa couche intermédiaire) avec une descente de gradient stochastique. Les résultats suivants sont obtenus :

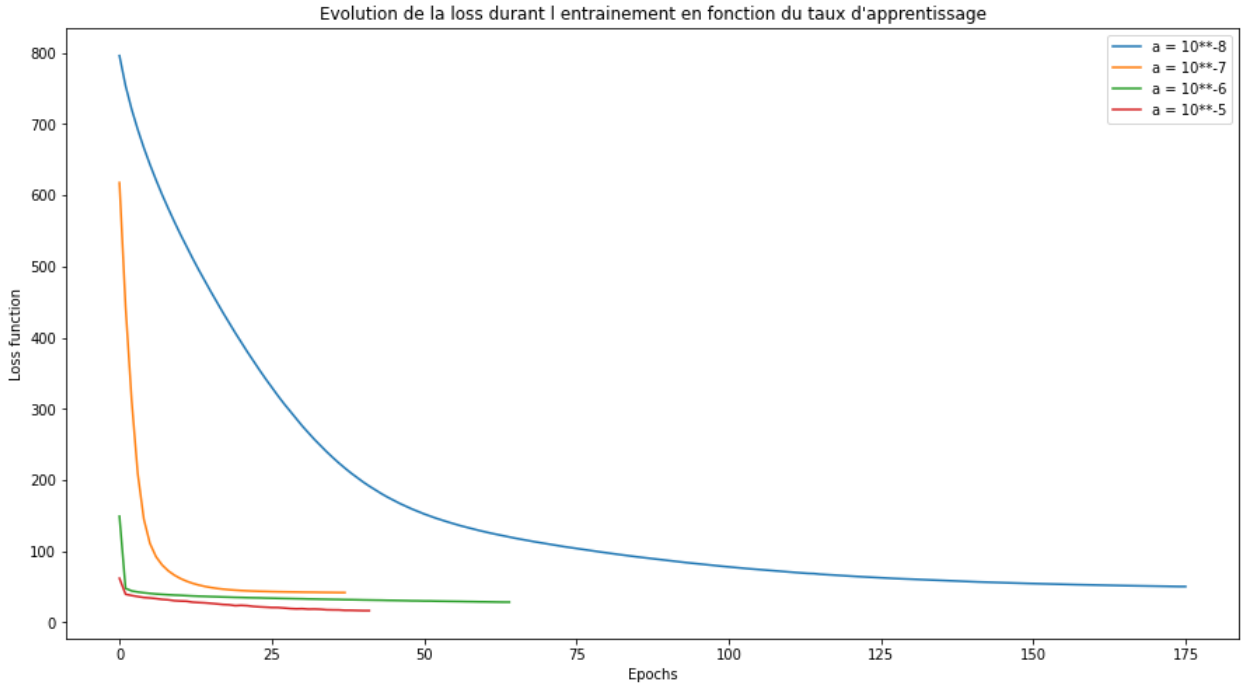


FIGURE 4.19 – Comme nous pouvons le constater, un petit taux d'apprentissage (en bleu) requiert énormément de temps lors de la phase d'entraînement. Par contre, cette fois-ci, un taux d'apprentissage plus grand que $\alpha = 10^{-5}$ ne permet pas de converger.

Sur la Figure ci-dessus, on constate qu'un taux d'apprentissage petit ($\alpha = 10^{-8}$) requiert davantage d'itérations qu'un taux d'apprentissage plus grand ($\alpha = 10^{-5}$), tel que attendu. Cependant, on constate qu'une descente de gradient stochastique requiert un taux d'apprentissage plus petit que celui utilisé par la méthode d'Adam. En effet, lors de la rétropropagation du gradient, une descente de gradient stochastique va mettre à jour les poids avec un pas constant dicté par le taux d'apprentissage. Or, au fur et à mesure des itérations, le gradient va diminuer. Par

conséquent, si on choisit un taux d'apprentissage trop grand, le modèle va finir par ne pas converger. Ce problème ne se pose cependant pas pour la méthode d'Adam. En effet, à chaque itération, elle utilise un pas adaptatif qui va dépendre du gradient. Toutefois, comme nous l'avons constaté à la Figure 4.18, un taux d'apprentissage trop grand peut faire converger le modèle vers un mauvais optimum local.

Finalement, nous pouvons conclure que la méthode la plus optimale pour notre problème est celle d'Adam avec un taux d'apprentissage $\alpha = 10^{-3}$. En effet, on constate qu'elle permet d'obtenir de meilleurs performances et ce, plus rapidement, que lors d'une descente de gradient stochastique.

4.2.7 Influence de la taille des batchs

Pour rappel, un batch correspond à un sous-ensemble des données. Par exemple, prenons le cas d'un batch de taille 32. Cela signifie que l'on va prendre 32 échantillons du *training set* et, seulement ensuite, entraîner le réseau. La seconde itération consistera à utiliser 32 autres échantillons. Dès lors, à chaque itération, le réseau est entraîné sur ce sous-ensemble de données. Sur la Figure suivante, on montre les performances du réseau lors de la phase de test en fonction de différentes tailles de batch.

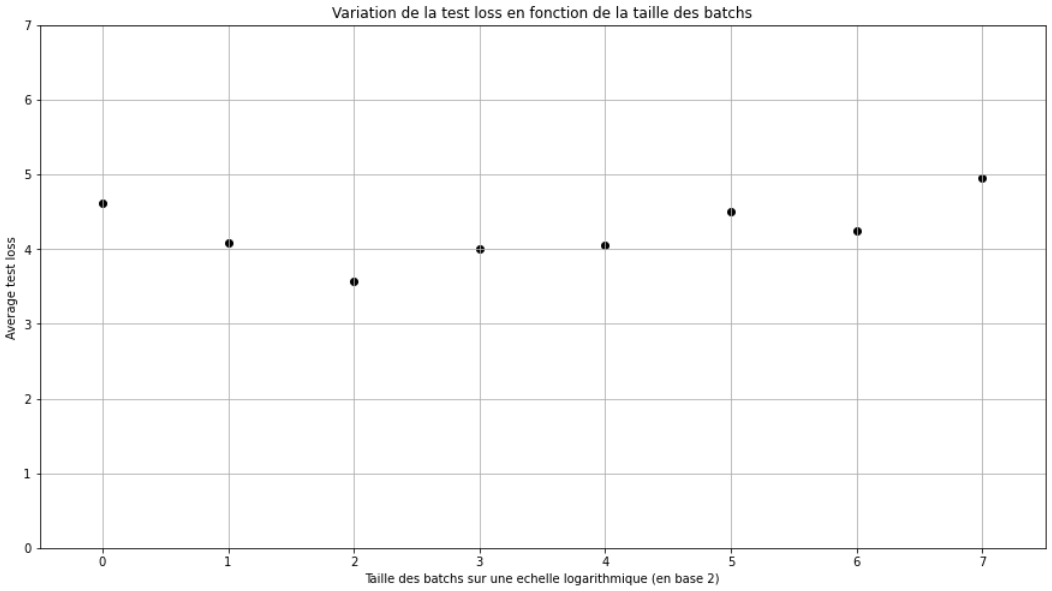
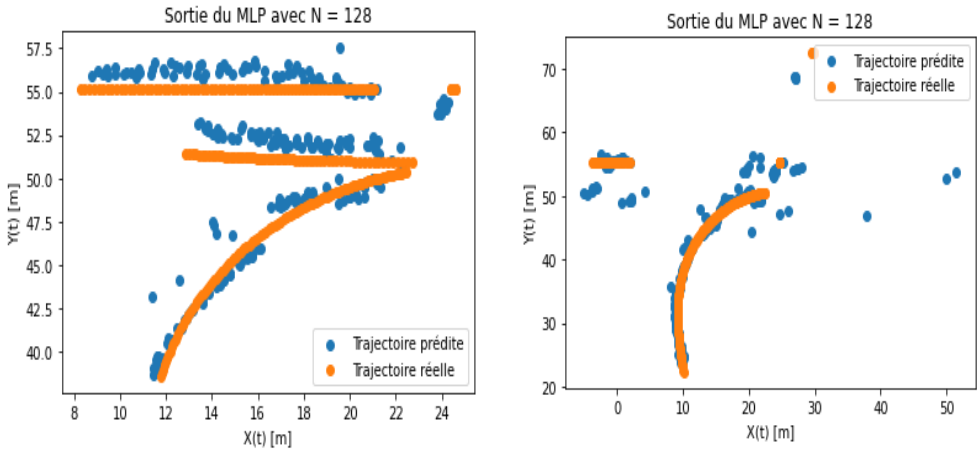


FIGURE 4.20 – Globalement, on constate qu'au plus la taille des batchs est grande, au plus les performances du réseau sont mauvaises.

Sur la Figure ci-dessus, on peut remarquer qu'utiliser des batchs permet d'améliorer les performances du réseau. En effet, pour une taille de batch étant égale à 1 ou à 2, les performances sont moins bonnes que celles obtenues avec une taille de batch égale à 4. Par ailleurs, on constate que les performances décroissent à mesure que la taille des batchs augmente. Ceci peut s'expliquer par le fait que le réseau n'a plus suffisamment de données sur lesquelles réaliser une itération. Par conséquent, la mise à jour des poids n'est pas suffisamment efficace. Par ailleurs, il est bon de noter que la taille optimale des batchs devrait dépendre du nombre de données disponibles. Intuitivement, utiliser un batch de taille 32 lorsque l'on dispose de 1000 données d'entraînement ou bien de 10000 n'est pas la même chose.

Finalement, on montre des résultats de prédiction dans le cas d'une taille de batch optimale, c'est-à-dire une taille de 4.



(a) (b)

FIGURE 4.21 – Ces deux Figures illustrent des prédictions de trajectoire pour une taille de batch égal à 4. On constate que les performances du réseau sont plutôt bonnes.

4.3 Transformer

Un Transformer est une architecture récente développée en 2017 qui utilise un mécanisme d'attention permettant de déterminer sur quelle partie de l'entrée il faut se concentrer. Ce réseau fournit de très bons résultats en traitement naturel du langage (Natural Language Processing) et, depuis peu, a été utilisé pour réaliser de la prédiction de trajectoire de piétons ([26]) ou de véhicules ([27]). Dans ce qui suit, l'architecture du réseau va être présentée et l'influence des différents paramètres sera ensuite discutée. Finalement, des résultats de prédictions de trajectoire par le Transformer seront montrés. A la Section 4.4, nous comparerons les résultats fournis par le filtre de Kalman, le perceptron multi-couches et le Transformer. Nous conclurons alors quant à l'architecture qui semble la plus adaptée pour ce travail.

Par ailleurs, il est bon de rappeler que ce travail se base sur un système bimodal radar-caméra. Grâce aux résultats obtenus dans [28], le choix de l'utilisation d'un Transformer s'est avérée prometteuse. En effet, ce papier scientifique traite de l'utilisation d'un transformer multimodal en traitement naturel du langage et consiste à dédier un Transformer par modalité. Nous reprenons donc cette idée et l'appliquons au cas d'un système radar-caméra. Cependant, étant donné la faible dimension des entrées, il nous semblait plus approprié de n'utiliser qu'un seul Transformer plutôt que deux.

4.3.1 Architecture du Transformer monomodal

Un Transformer, tel que présenté dans [29], est un réseau constitué de l'architecture suivante :

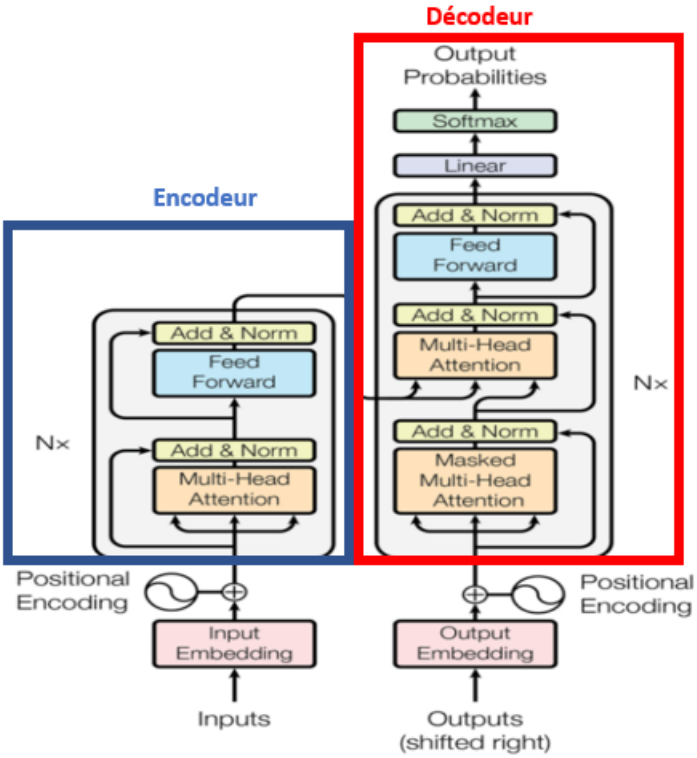


FIGURE 4.22 – Globalement, un transformer est constitué de deux parties : un encodeur et un décodeur. Chacune de ces parties est constituée de N couches (en général $N = 6$), elles-mêmes faites de sous-couches.

Sur la Figure ci-dessus, on constate qu'une architecture transformer comporte deux parties : un encodeur et un décodeur. La première partie est responsable de fournir une représentation abstraite de l'entrée tandis que la deuxième partie est responsable de la prédiction d'une séquence. Bien que ces deux rôles soient différents, on peut constater que l'encodeur et le décodeur partagent à peu de chose près les mêmes modules. Ils sont tous les deux faits de N couches. Dans une couche de l'encodeur, on retrouve un module *Multi-Head Attention* et un module *Feed Forward*, tout comme dans une couche du décodeur. Par contre, cette dernière partie comporte également une couche *Masked Multi-Head Attention*, ce qui la différencie de la partie servant à l'encodage. En outre, on peut remarquer que chaque module est suivi d'une couche de normalisation (*Add and Norm*) et qu'il y a des connexions résiduelles entre deux sous-modules. Celles-ci, tel qu'expliqué dans [30], permettent un meilleur entraînement du réseau. Finalement, on peut remarquer la présence d'un module *positional encoding* à l'entrée de l'encodeur et du décodeur.

Multi-Head Attention

Le mécanisme d'attention est un élément clef de cette nouvelle architecture. Pour expliquer son fonctionnement, nous allons l'appliquer en utilisant des mots comme en traitement naturel du langage. Pour ce faire, on suppose qu'un mot X_i est de dimension d et qu'il y a n mots. On peut donc former une matrice X de dimensions $n \times d$. A partir de cette matrice, l'idée est de calculer trois représentations : Q (Query), K (Key), V (Value). Celles-ci sont données par :

$$Q = XW^Q \tag{4.44}$$

$$K = XW^K \tag{4.45}$$

$$V = XW^V \tag{4.46}$$

Où les matrices W^Q, W^K, W^V sont des matrices déterminées lors de la phase d'entraînement du réseau et de dimensions respectives $d \times d_k$, $d \times d_k$ et $d \times d_v$.

Le processus d'attention va alors calculer la similarité entre deux mots en se basant sur le fait que deux mots ayant des caractéristiques similaires doivent avoir une valeur proche. Pour ce faire, un vecteur v de V représente la valeur du mot, un vecteur k de K représente ses caractéristiques et un vecteur q de Q représente un mot pour lequel on doit déterminer sa valeur. Autrement dit, suite à la phase d'entraînement du réseau, le transformer a pu déterminer les matrices W^Q, W^K, W^V. Dès lors, pour une certaine entrée X formée de mots, le mécanisme d'attention va permettre de déterminer sur quelles parties il faut se concentrer, c'est-à-dire à quelles parties il faut accorder davantage d'attention. A la suite de ce mécanisme, on a donc déterminé une abstraction Z_i pour chaque mot X_i comme illustré à la Figure suivante :

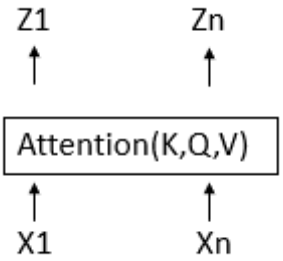


FIGURE 4.23 – Cette Figure représente le mécanisme d'attention. On constate que chaque abstraction Z_i est calculée de façon indépendante en traitant tous les mots X_i en parallèle.

Mathématiquement, connaissant les représentations Q, K, V de la matrice X,

on définit l'attention de la façon suivante :

$$Attention(K, Q, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{4.47}$$

En pratique, un transformer est composé d'une couche multi-head attention qui met en parallèle h mécanismes d'attention. Ainsi, pour un même mot X_i, plusieurs abstractions Z_i sont créées. La sortie de cette couche correspond alors à la concaténation de chacune des abstractions. Autrement dit,

$$MHA(K, Q, V) = Concat(head_1, \ldots, head_h)W^0 \tag{4.48}$$

où $head_i = Attention(KW_i^K, QW_i^Q, VW_i^V)$.

Masked Multi-Head Attention

Le module *Masked Multi-Head Attention* fonctionne comme le module *Multi-Head Attention* sauf que l'on masque certaines entrées. En effet, le décodeur a pour but de prédire une séquence (une phrase) faite d'un certain nombre de mots. Or, la prédiction du mot Y_i ne doit dépendre que des mots déjà prédits par le passé et non pas des mots futurs. Ainsi, pour permettre un processus autorégressif, un phénomène de masking doit être mis en place. Ceci est illustré à la Figure suivante provenant de [31] :

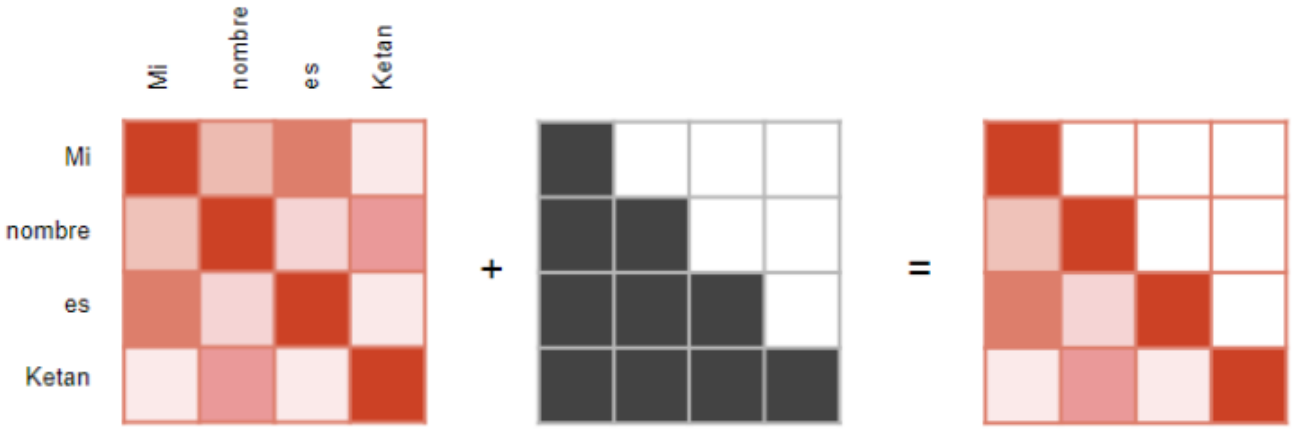


FIGURE 4.24 – Comme nous pouvons le constater, le masking permet un processus autorégressif de telle sorte que la sortie Y_i ne dépende que du passé.

En pratique, les mots masqués reçoivent un score d'attention valant $-\infty$. De cette manière, la fonction softmax transformera cette valeur en zéro, signifiant qu'il ne faut pas prêter attention à ces mots-là.

Feed Forward

Le module *feed forward* correspond à un réseau de neurones de type *fully connected*, à l'instar du perceptron multi-couches. Pour rappel, l'architecture d'un tel réseau est expliquée à la Section 4.2. L'utilisation d'une telle sous-couche a plusieurs avantages. D'une part, la sous-couche *Multi-Head Attention* permet de faire de l'extraction de caractéristiques des données, ce qui correspond à des opérations locales. En utilisant un réseau *fully connected* de type MLP à la suite de ces opérations, la sortie de chaque neurone d'une couche sera l'entrée de tous les neurones de la couche suivante. Dès lors, ceci permettra de globaliser les résultats. Ainsi, la sortie de l'encodeur dépendra de tout ce qui a été mis à son entrée et non pas seulement d'un sous-ensemble de l'entrée. D'autre part, le MLP va également pouvoir réaliser une classification sur base des caractéristiques extraites à la suite du module *Multi-Head Attention*.

Add and norm

Le sous-couche *Add and norm* constitue une couche de normalisation et son implémentation pratique est expliquée dans [32]. Concrètement, elle consiste à normaliser tous les neurones d'une même couche avec la même moyenne et la même variance. Ceci permet de ne pas avoir de contrainte sur la taille des mini-batchs. Elle permet également de faire passer les composantes résiduelles des couches précédentes ce qui, dans le cadre de grands réseaux de neurones comme les transformers, permet de ne pas perdre d'information dans le cas où la couche antérieure au *Add and Norm* fournisse des abstractions erronées ou mauvaises.

Positional encoding

Dans une phrase, l'ordre des mots est important. Sans cet ordre, la phrase n'a plus de sens. Or, tant dans l'encodeur que dans le décodeur, l'ordre des mots n'est pas pris en compte. C'est pourquoi, l'utilisation d'un module *positional encoding* permet de tenir compte de l'ordre des mots dans la phrase. Concrètement, pour un mot à la position t, le module fournit un vecteur p_t de dimensions d tel que :

$$p_t^i = \begin{cases} \sin(\omega_k t) \text{ si } i = 2k \\ \cos(\omega_k t) \text{si } i = 2k+1 \end{cases} \tag{4.49}$$

où la fréquence ω_k est définie par $\omega_k = \frac{1}{10000^{\frac{2k}{d}}}$.

Un avantage de ce module est que, pour chaque mot, il fournit un encodage unique, déterministe et borné de la position.

4.3.2 Paramètres utilisés

Au vu de ce qui a été montré pour un réseau de type MLP, nous pouvons supposer que le passé aura une influence positive sur les résultats et que la méthode d'Adam fournira de bons résultats. Dans cette section, nous allons donc étudier l'impact des paramètres propres à l'architecture du transformer : le nombre de couches utilisées, le nombre de neurones formant la couche intermédiaire du MLP ainsi que le nombre d'abstractions fournies par le module *Multi-Head Attention*. Par ailleurs, comme expliqué à la Section 4.3.1, un transformer est un réseau de neurones fait d'un encodeur et d'un décodeur. Cependant, cette deuxième partie est utilisée pour prédire une séquence, ce qui n'est pas l'objet de ce travail. En effet, nous nous intéressons à une application de filtrage des données. Par conséquent, seule la partie relative à l'encodeur sera étudiée en détails. En outre, nous n'utiliserons pas de module *positional encoding* car l'ordre des données est déjà pris en compte lorsque nous fournissons une séquence temporelle au transformer. Dans ce qui suit, le réseau sera optimisé en fonction de l'erreur quadratique moyenne.

Nombre de neurones présents

Comme expliqué à la Section 4.3.1, tant l'encodeur que le décodeur se terminent par un réseau neuronales complètement connectés tel un perceptron multi-couches. Par la suite, nous considérons ce réseau de la façon traditionnelle c'est-à-dire avec une couche intermédiaire. Dans cette section, nous allons déterminer l'influence du nombre de neurones de cette couche. La Figure suivante représente les performances du réseau lors de la phase de test. Le transformer a été entraîné avec un batch de taille 8 et la méthode d'Adam (avec un taux d'apprentissage $\alpha = 10^{-3}$). Le module *Multi-Head Attention* fournit huit abstractions.

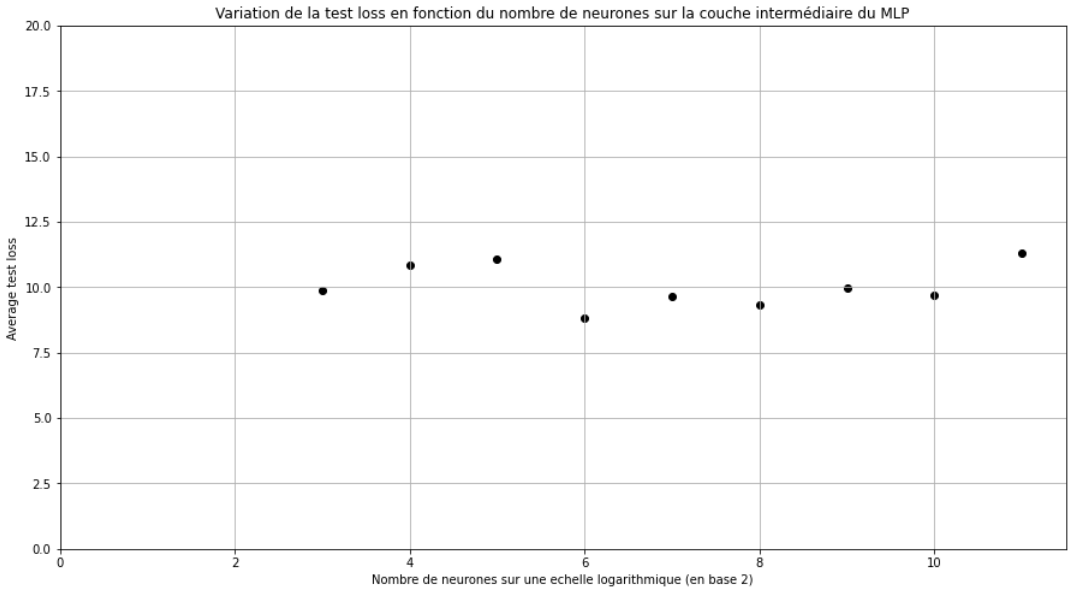
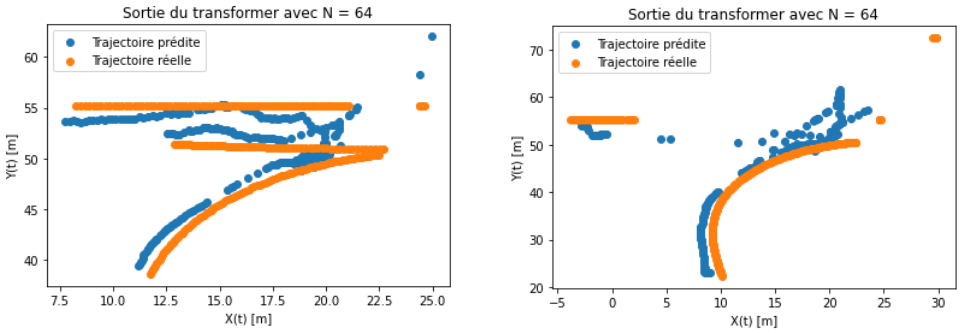


FIGURE 4.25 – Comme nous pouvons le constater, les meilleures performances sont obtenues pour un nombre de neurones $N = 64$. Globalement, on constate également que le nombre de neurones n'influence pas beaucoup les performances du transformer.

Comme nous pouvons le constater sur la Figure ci-dessus, le même genre de résultats est obtenu que ceux illustrés aux Figures 4.10, 4.11 et 4.12 pour le perceptron multi-couches. Les mêmes commentaires peuvent donc s'y appliquer. Cependant, une différence notable se fait au niveau des trajectoires prédites. Celle-ci est illustrée à la Figure suivante.

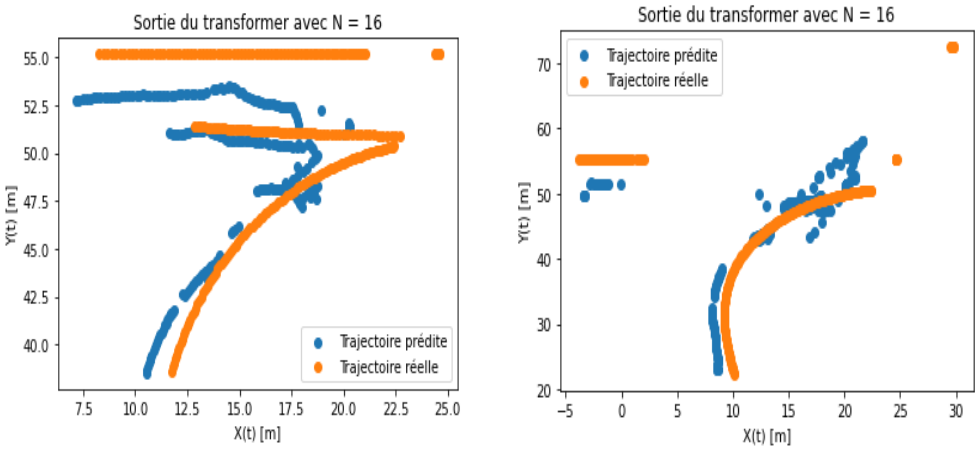


(a) (b)

FIGURE 4.26 – Ces deux Figures illustrent des prédictions de trajectoire avec un transformer

Comme illustré sur la Figure ci-dessus, les trajectoires prédites par le transformer ne correspondent pas à un amas de points bruités. La trajectoire semble beaucoup plus réaliste, plus stable, plus lisse et moins bruitée. Il s'agit d'un continuum de points et non plus d'un amas de points. Ainsi, on peut supposer que le transformer est beaucoup plus à même d'apprendre la dynamique et la cinématique des véhicules. Par ailleurs, on constate très peu d'outliers, c'est-à-dire de points très mal prédits.

Une autre caractéristique du transformer est qu'il est également robuste et fournit des résultats stables lorsque l'on change le nombre de neurones de la couche intermédiaire du MLP. Ceci est pratique pour réduire le temps d'entraînement du réseau. Par contre, bien qu'il s'agit d'une dynamique et d'une cinématique stables, les trajectoires prédites semblent plus éloignées des trajectoires réelles, contrairement aux résultats obtenus avec un MLP (voir Figure 4.16).



(a) (b)

FIGURE 4.27 – On peut constater que la dynamique et la cinématique prédites des véhicules est stable mais semblent plus éloignées que le cas réel.

Influence du taux d'apprentissage

Précédemment, nous avons vu que le taux d'apprentissage était un paramètre important dans un réseau neuronal de type MLP. En effet, lorsqu'il est trop petit, le réseau prend trop de temps pour converger et lorsqu'il est trop grand, il risque de converger vers un mauvais optimum. Dans cette section, nous allons brièvement regarder l'implication de ce paramètre pour un réseau de type transformer et ce, avec la méthode d'Adam. Ceci est illustré à la Figure suivante :

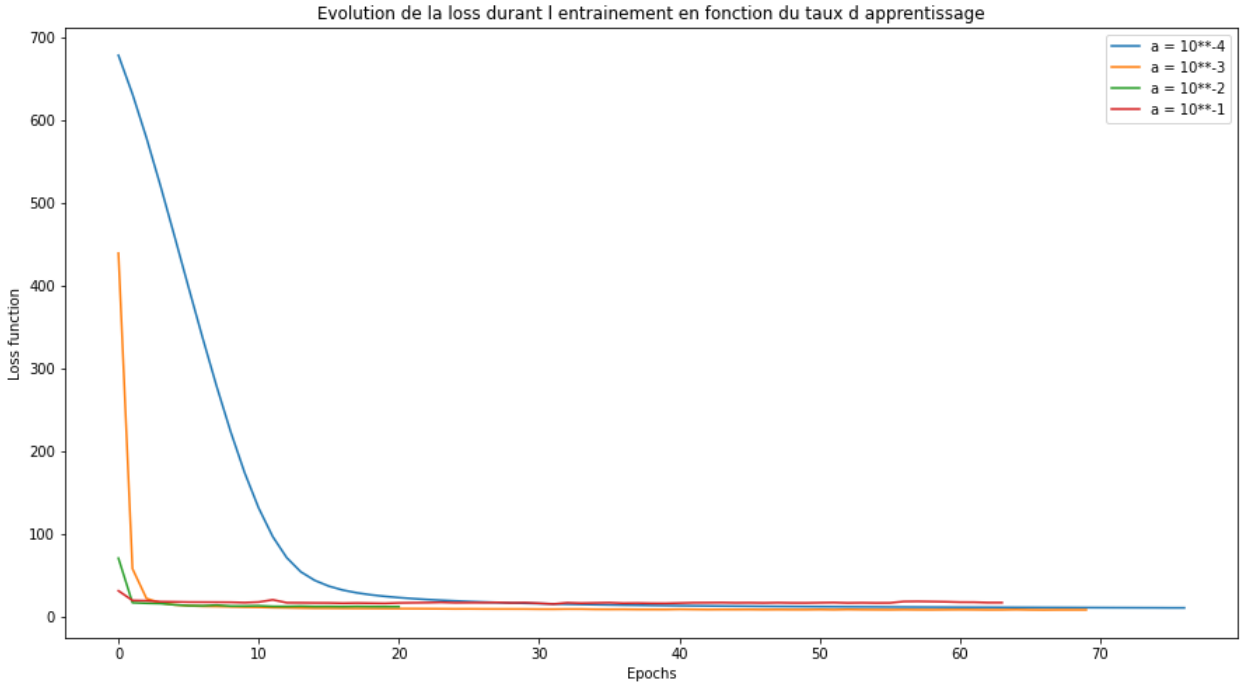


FIGURE 4.28 – On constate que le transformer converge, que l'on utilise un grand taux d'apprentissage ($\alpha = 10^{-1}$) ou un petit taux d'apprentissage ($\alpha = 10^{-4}$).

Ainsi, comme observé sur la Figure ci-dessus, un petit taux d'apprentissage requiert davantage d'itérations pour que le modèle converge. Cependant, une différence notable avec le réseau neuronale MLP est que le transformer converge correctement pour un grand taux d'apprentissage ($\alpha = 10^{-1}$). Intuitivement, ceci peut s'expliquer par le fait que le réseau est plus grand. En effet, en général, un grand réseau (en terme de nombre de couches et de sous-couches) est plus robuste. Cependant, il existe une taille optimale telle qu'un réseau encore plus grand arrêtera de fournir de meilleures performances.

Sur la Figure 4.28, on peut cependant remarquer que le plus grand taux d'apprentissage (en rouge) commence à fournir de moins bons résultats. En effet, le modèle converge avec une fonction coût un peu plus élevée que pour les autres taux d'apprentissage. Ceci peut également s'observer par le fait que, aux alentours de la dixième epoch, la fonction coût augmente subitement.

Influence de la taille des batchs

La taille des batchs utilisés lors de la phase d'entraînement d'un réseau de type Transformer est un paramètre important. En effet, au plus la taille est grande (64, 128, ...), au plus le transformer va utiliser de données à chaque itération lors de l'entraînement. Ainsi, cette phase ira plus vite. Contrairement à un réseau simple de type MLP, le transformer est plus grand et prend donc plus de temps à entraîner. Par conséquent, la taille des batchs est un paramètre à optimiser de façon à obtenir un réseau entraîné plus rapidement. La Figure suivante compare les performances du transformer pour différentes tailles de batch :

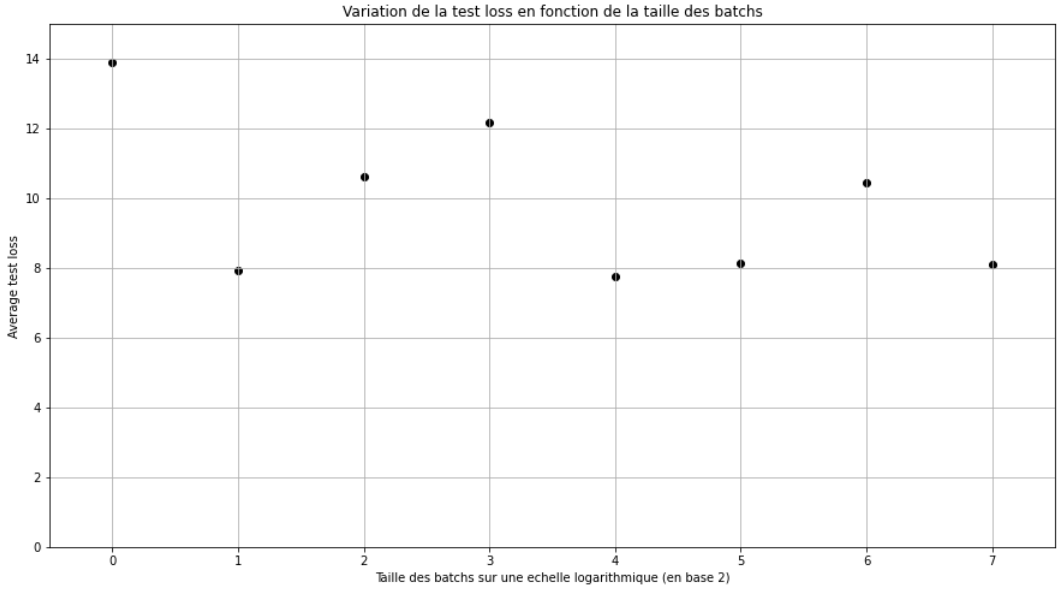


FIGURE 4.29 – Contrairement à un MLP, on constate que les performances d'un transformer ne diminue pas toujours en fonction de la taille des batchs. On constate qu'un batch de taille 16 fournit les meilleures performances.

Sur la Figure ci-dessus, on constate que les performances optimales sont obtenues pour un batch de taille 16 ou bien un batch de taille 2. En pratique, il vaut mieux utiliser un batch de taille 16 étant donné que le temps d'entraînement du réseau sera plus court.

Influence du nombre de couches de l'encodeur

L'encodeur d'un transformer est fait de deux modules principaux : le *Multi-Head Attention* et le MLP. La taille de cette partie de l'architecture est alors dictée par le nombre de couches utilisées. En traitement naturel du langage (NLP), il est courant

d'utiliser une taille $N = 6$. Dans cette section, nous proposons d'optimiser ce paramètre de façon à déterminer s'il faut réellement entraîner un aussi gros réseau ou si l'on peut se permettre d'utiliser une plus petite architecture. Les résultats suivant sont obtenus :

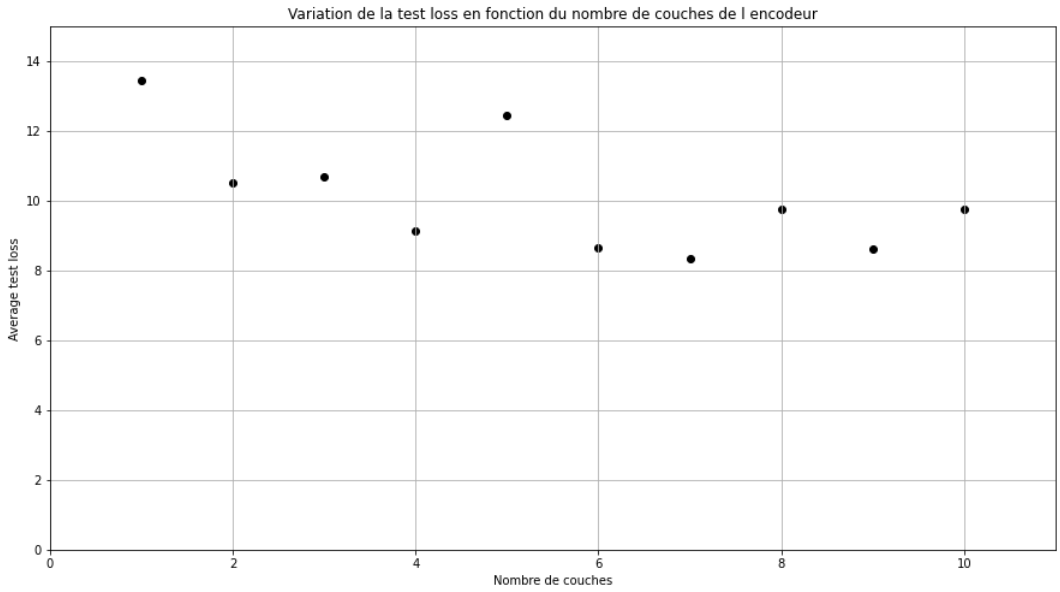


FIGURE 4.30 – On constate que le cas optimal est $N = 7$ mais que $N = 6$ est tout aussi bon. Toutefois, on pourrait se permettre d'utiliser quatre couches étant donné que cela fournit des performances comparables.

Ainsi, bien que le traitement naturel du langage et la prédiction de trajectoire soient deux domaines complètement différents, on constate que six couches permettent de fournir les meilleurs résultats. De plus, on constate qu'un nombre de couches plus grand n'apporte rien étant donné que les performances du transformer sont moins bonnes. On peut supposer que ceci est dû à un réseau trop grand et à un manque de données. Dans ce cas, le transformer a plus de chance de faire du sur-entraînement. Par ailleurs, une observation intéressante est qu'un petit nombre de couches ($N = 2$, $N = 3$) fournit des résultats corrects. Par conséquent, pour l'utilisation d'un transformer pour lequel un faible temps d'entraînement est requis, il vaudra mieux utiliser ce faible nombre de couches.

Influence du nombre d'abstraction du module *Multi-Head Attention*

Un module propre au transformer est celui relatif au *Multi-Head Attention*. Celui-ci permet de fournir, pour chaque entrée, un certain nombre d'abstractions. Chacune regroupant une caractéristique particulière. Il est donc intéressant de

déterminer si ce paramètre influence les performances du réseau. Ceci est illustré à la Figure suivante :

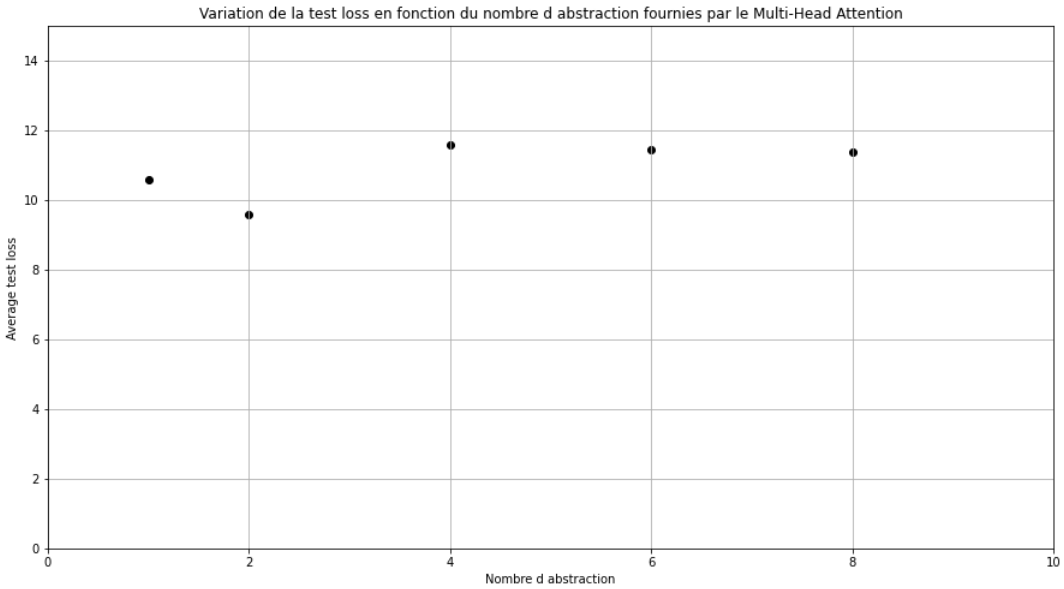


FIGURE 4.31 – On constate que ce paramètre n'influence pas beaucoup les performances du réseau étant donné que celles-ci sont quasiment constantes.

De part sa faible influence, il n'est donc pas nécessaire d'optimiser ce paramètre. Par la suite, nous prendrons un nombre d'abstraction égal à 8. Il s'agit en effet d'une valeur courante en NLP.

4.4 Comparaison des résultats

Cette section est consacrée à la comparaison et discussion des résultats obtenus pour les trois architectures présentées dans ce chapitre. On va notamment s'intéresser à trois caractéristiques :

1. la taille du modèle : l'espace de stockage nécessaire pour stocker les poids entraînés des modèles ;
2. le temps d'exécution : le temps que prend l'architecture pour filtrer N valeurs (réalisé sur les mêmes ressources calculatoires) ;
3. l'influence du passé : le nombre d'instants donnés dans le passé ;
4. la MSE : l'erreur quadratique moyenne entre la prédiction et la valeur réelle.

Le tableau 4.3 permet de comparer les résultats des différentes architectures présentées.

	Filtre de Kalman	MLP	Transformer
Espace de stockage [Kb]	0	16.6	267.91
$T_{exe}[s]$	3.14	0.9884	32.0971
Instants du passé donnés	1	8	8
MSE	55.05	3.47	8.79

TABLE 4.3 – Tableau récapitulatif des modèles utilisés

Ce tableau montre que les performances des architectures non-linéaires sont bien meilleures que les performances du filtre de Kalman, car leur MSE est largement inférieure à celle de l'architecture linéaire. Cette différence vient du fait que les deux réseaux de neurones sont entraînés sur des données synthétiques afin de filtrer les données de telle sorte que l'erreur quadratique soit minimale. Bien que les paramètres du filtre de Kalman soit optimisés, ils ne permettent pas la même flexibilité que les modèles non-linéaires. En effet, lors de cas particuliers où l'une des mesures prises par l'un des senseurs est incorrecte, la stratégie de fusion des mesures ne permet pas de faire face à cette valeur aberrante, créant une nouvelle piste tout en rendant la piste à laquelle elle devrait être associée moins précise. Cela entraîne l'augmentation de la MSE par rapport aux méthodes non-linéaires. On constate aussi que le temps d'exécution du MLP est inférieur à celui du filtre de Kalman, montrant ainsi les limites du filtre de Kalman sur les autres modèles.

Par rapport aux deux architectures non-linéaires, on remarque que la MSE du du MLP est meilleure que celle du Transformer, ce qui signifie que les prédictions du MLP se rapprochent plus des valeurs réelles. En analysant visuellement ces prédictions (Figure 4.21), on se rend compte que ces valeurs, bien que plus proches, sont très bruitées et ne suivent pas de trajectoires visibles. Ceci correspond plutôt à un amas de points qui se suivent approximativement. Cela montre que le MLP ne tient pas compte des instants du passé et donc, de la cinématique des cibles, et ne se contente uniquement que de restituer un point. Le Transformer, bien qu'ayant une MSE moins bonne mais tout à fait raisonnable, tient bien compte des instants précédents. En effet si on prend comme exemple les Figures 4.27, les trajectoires fournies se suivent sans s'éparpiller dans l'espace se rapprochant fortement des résultats fournis par le filtre de Kalman. Ceci est dû au mécanisme d'attention. Ce niveau d'abstraction supplémentaire engendre une perte en précision tout en "stabilisant" la prédiction créant des prédictions plus lisse. Ce lissage a cependant un coût en terme de temps d'exécution. En effet, il prend 32 fois plus de temps pour effectuer le même nombre de prédictions que le MLP, sans compter que la taille du modèle est 16 fois plus importante.

On en conclut que, bien que le MLP semble être le meilleur choix en terme de performances, le Transformer permet d'obtenir des trajectoires plus lisses telles que celles obtenues avec un filtre de Kalman mais avec une meilleure précision.

Chapitre 5

Validation expérimentale

Dans ce chapitre, les algorithmes utilisés et expliqués précédemment sont appliqués à des données réelles. Dans un premier temps, le hardware utilisé est brièvement présenté. Dans un second temps, les résultats pour chacune des modalités sont montrés. Finalement, les architectures expliquées précédemment sont appliquées sur ces données réelles. Plus précisément, elles sont entraînées sur les données simulées et sont ensuite appliquées sur des données réelles. L'équipement utilisé ainsi que la méthode de capture lors de la validation expérimentale a été développée par le précédent mémorant Alexis Duflot, auteur de [7]. Ces algorithmes seront supposés acquis par la suite.

5.1 Fiche technique du hardware utilisé

Les Tables 5.1 et 5.2 reprennent les fiches techniques du radar et de la caméra utilisés ainsi que les paramètres utilisés lors de la validation expérimentale.

Radar K-MD2-RFB-01F-01	
Résolution	256 × 256 (vitesse × distance)
Images par secondes	20 [fps]
Gain de l'émetteur	24 [GHZ]
Portée	0-250 [m], 1 [m] de résolution
Intervalle de vitesse possible détecté	$\pm$130 $[km/h^{-1}]$, 1 $[km/h^{-1}]$ de résolution
Angle d'ouverture en élévation	18.2 [°], 0.1 [°] de résolution
Angle d'ouverture en azimut	32.8 [°], 0.1 [°] de résolution
Interface	Ethernet ou Connections série
Alimentation	10-16 [V] / 600 [m A]
Dimensions	120 × 72 × 15 [mm]
Distance horizontale entre antenne d_x	21.8 [mm]
Distance verticale entre antenne d_z	39.6 [mm]
Paramètres utilisés	
Bande passante de rampe B	554 [MHz]
Bande passante d'échantillonage B_s	545.5[MHz]
Fréquence de départ f_{dep}	23.8 [GHz]
Échantillons utilisés N	256
Résolution en distance ΔR	0.274 [m]
Distance maximale R_{max}	70.122 [m]
Résolution en vitesse Δv	0.175 [m/s]
Vitesse absolue maximale v_{max}	22.22 [m/s]
Ambiguïté d'élévation v_{amb}	18.2 [°]
Ambiguïté azimutale u_{amb}	32.8 [°]

TABLE 5.1 – Fiche technique du radar [19]

Caméra ELP 1/3" CMOS AR0330	
Résolution	320 × 180, 320 × 240, 640 × 360, 640 × 480, 1280 × 720, 1920 × 1080
Format vidéo	H.264/MJPEG/YUY2(YUYV)
Images par secondes	30 fps, 25 fps, 20 fps, 10 fps, 5 fps
Exposition automatique	Oui
Interface	USB 2.0
Alimentation	5 [V] via USB / 150 [m A]
Lentille	12 [mm]
Angle d'ouverture	28 [°]
Distance focale	5695.8[pixel]
Erreur moyenne de re-projection	0.78 [pixels]
Dimensions	38 × 38 × 32 [mm]
Paramètres utilisées dans le cadre du mémoire	
Résolution	1280 x 720 [pixel]
Images par seconde	20 [fps]

TABLE 5.2 – Fiche technique de la caméra provenant de [16] et [7]

5.2 Données du radar et comparaison avec la simulation

Au Chapitre 1, nous présentions la simulation d'un radar Doppler. Dans cette section, nous allons confronter cette simulation avec un radar réel. Pour rappel, un radar permet de déterminer la position et la vitesse d'une cible à l'aide de deux cartes thermiques.

5.2.1 Calibration

Sur la Figure 5.1, on observe une carte thermique distance-vitesse sans calibration où l'on constate plusieurs non-idéalités.

— Une droite verticale axée en 0. Elle provient du traitement du signal à l'aide de la transformée de Fourier (du décalage de la fréquence) .
— Deux amas de points "en miroir" opposés qui sont deux non-idéalités dûes aux composants physiques du hardware.

Pour enlever ces non-idéalités, il faut filtrer les signaux analysés en enlevant l'arrière-plan bruité. Pour calculer cet arrière-plan, il a été décidé de prendre trois images successives ne contenant aucun véhicule, d'additionner leurs valeurs

échantillonnées et de moyenner le tout. Le résultat obtenu est illustré sur la Figure 5.2a.

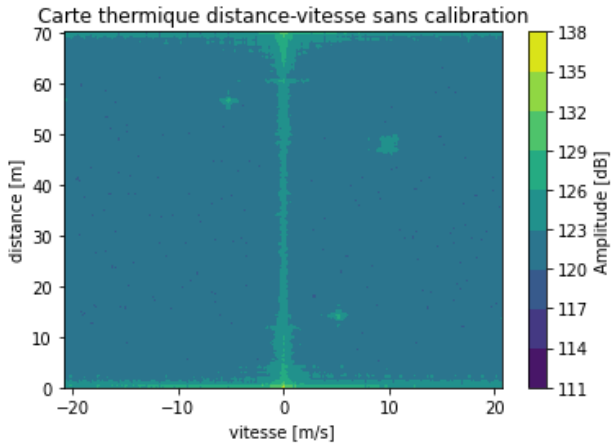
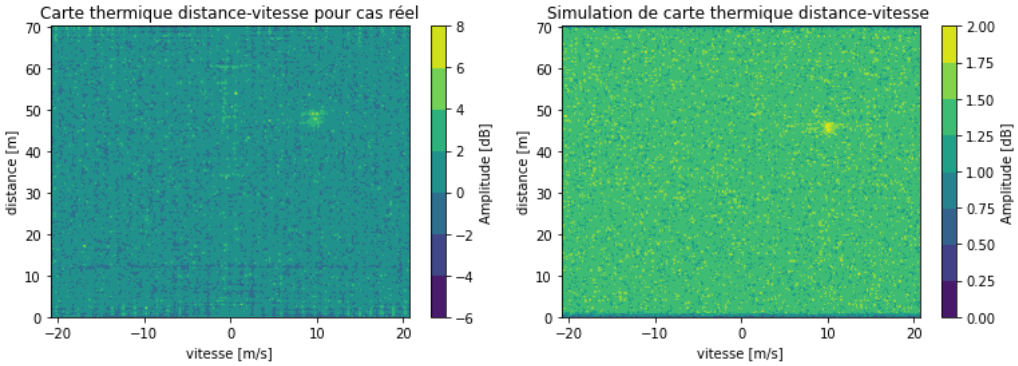


FIGURE 5.1 – Carte thermique distance-vitesse sans calibration illustrant les non-idéalités. Le seul véhicule détecté se trouve à une distance d'environ 50 [m] du radar et s'en éloigne à une vitesse de 10 [m/s].

5.2.2 Comparaison avec la simulation

Comme expliqué au Chapitre 1, la position des cibles détectées par le radar se situe sur des heatmaps. La Figure suivante compare une heatmap réelle et une heatmap simulée.



(a) Validation expérimentale pour un véhicule se situant à 47 [m] et roulant à 10 [m/s]

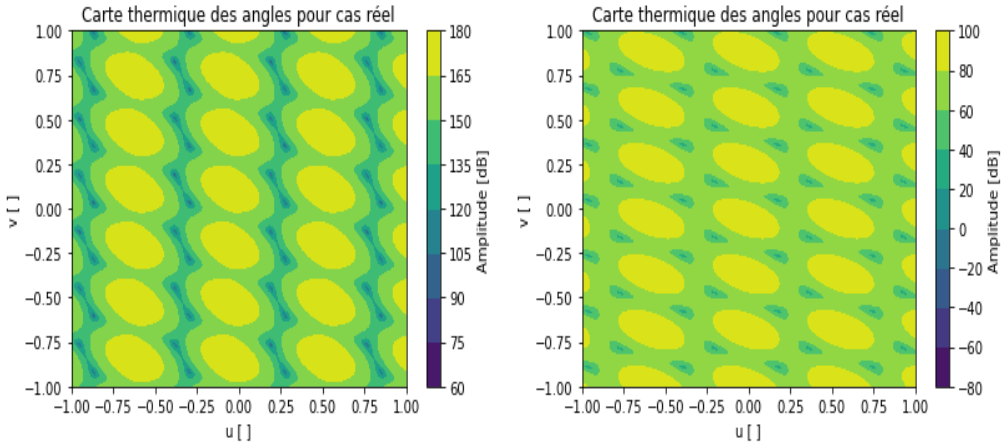
(b) Simulation d'un véhicule se situant à 47 [m] et roulant à 10 [m/s]

FIGURE 5.2 – Comparaison entre une carte thermique distance-vitesse réelle (à gauche) et une carte thermique distance-vitesse simulée.

La Figure ci-dessus permet de valider la simulation du radar. En effet, on y retrouve clairement un amas de points représentant un véhicule détecté par le radar. On y observe également le spectre micro-Doppler : celui-ci se caractérisant par un étalement sur l'axe des vitesses. On observe également un étalement le long de l'axe de la distance : celui-ci correspond à la distance inter-essieu des roues du véhicule. Ainsi, globalement, la simulation radar modélise bien un véhicule.

Une différence provient de l'intensité du signal reçu. En effet, pour la simulation, il était fort difficile d'estimer l'intensité du champ électrique capté par le radar, celui-ci dépendant de beaucoup de facteurs comme l'environnement, la réflectivité des objets dans le champ de vue du radar, etc. En conséquence, il a été décidé de négliger l'intensité et de choisir une amplitude arbitraire de 1 pour l'intensité de chacune des cibles.

Par ailleurs, comme expliqué au Chapitre 1, la deuxième carte thermique fournie par un radar Doppler est celle illustrant les angles. La Figure suivante compare un cas réel et le même cas simulé.



(a) Validation expérimentale de la carte thermique pour un véhicule

(b) Simulation de la carte thermique d'un véhicule à une position particulière

FIGURE 5.3 – Comparaison de la carte thermique des angles pour le cas réel (à gauche) et simulé (à droite) d'un véhicule se trouvant à $\theta = 90[°]$ et $\phi = -3$ [°]

Sur la Figure 5.3, on remarque que l'allure des deux graphiques est identique. Seule l'intensité varie, pour la même raison que pour la carte thermique distance-vitesse. Cependant dans ce cas-ci, par l'Equation (3.28), l'amplitude choisie n'influence pas la recherche du "bon" maximum local permettant de lever les

ambiguïtés, étant donné que le signal reçu aux trois antennes doit théoriquement avoir la même amplitude mais déphasée.

5.2.3 Problèmes rencontrés

La fiche technique du radar indique que celui-ci renvoie un jeu d'échantillons (soit $3 \times 256 \times 256$ données) toutes les 0.05 secondes, chaque antenne recevant 256×256 échantillons. Cependant, en pratique, nous n'avons pas pu observer cela. En effet, nous étions plutôt de l'ordre d'un jeu de données toutes les 0.15 secondes, plusieurs parties des jeux de données n'étant pas enregistrées au bon moment. Nous avons ainsi pu remarquer que les informations enregistrées par le radar contenaient entre 0 et $5 \times 256 \times 256$ données pour chaque échantillon, ce qui laisse suggérer que le transfert et l'enregistrement des données (réalisés par Alexis Duflot) au niveau de la gestion du hardware pose problème. Par conséquent, un même véhicule ne sera capté que 5 ou 6 fois lors de son passage dans le champ de vision du radar. La trajectoire de ce véhicule perçue par ce senseur sera donc constituée de quelques points espacés de plusieurs mètres.

En conséquence, la levée des ambiguïtés du radar semble mal réalisée car les valeurs des angles trouvées comportent plus de bruit que ceux simulés, rendant le résultat moins précis. Ceci est dû au fait que les algorithmes utilisés pour extraire les données se basent sur l'hypothèse d'une synchronisation parfaite entre le radar et la caméra. Du fait que les échantillons du radar n'arrivent pas à intervalle régulier, la synchronisation n'est pas toujours parfaitement réalisée. Il en résulte qu'une position estimée par la caméra ne correspondra pas toujours à la même position estimée par le radar. Autrement dit, le délai dû à la mauvaise synchronisation entraîne que, pour lever les ambiguïtés du radar à un instant t, l'algorithme ne va pas utiliser la position équivalente estimée par la caméra. Ceci est illustré à la Figure suivante :

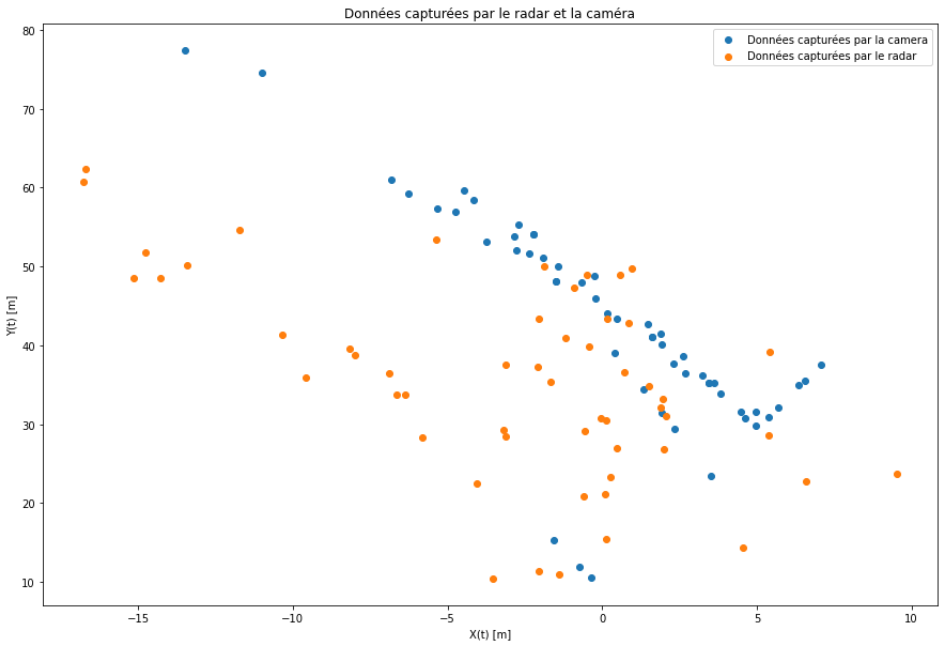


FIGURE 5.4 – On constate que les positions estimées par le radar sont extrêmement bruitées. Il faudra donc filtrer beaucoup plus ces données à l'aide des architectures.

5.3 Données de la caméra

Sur la Figure suivante, nous illustrons la capture des données par la caméra après traitement avec l'algorithme YOLO.

FIGURE 5.5 – On constate que l'algorithme YOLO, bien qu'il ne soit pas entraîné, capture correctement les véhicules.

Sur la Figure ci-dessus, on constate que l'algorithme YOLO permet de capturer correctement les véhicules. Une différence par rapport à la simulation est la présence de panneaux routiers dans l'environnement. Ceux-ci devront être supprimés par la suite pour ne traiter que les objets d'intérêt, c'est-à-dire les véhicules. Par ailleurs, nous pouvons constater que la bouding box n'épouse pas correctement le véhicule à cause de l'angle de vue de la caméra. Il en résulte que celui-ci est vu en biais et que donc, la détermination de sa dimension principale est ambiguë. Cependant, contrairement à ce que l'on pourrait croire, l'algorithme d'extraction des données de la caméra reste stable. Ceci est illustré à la Figure suivante :

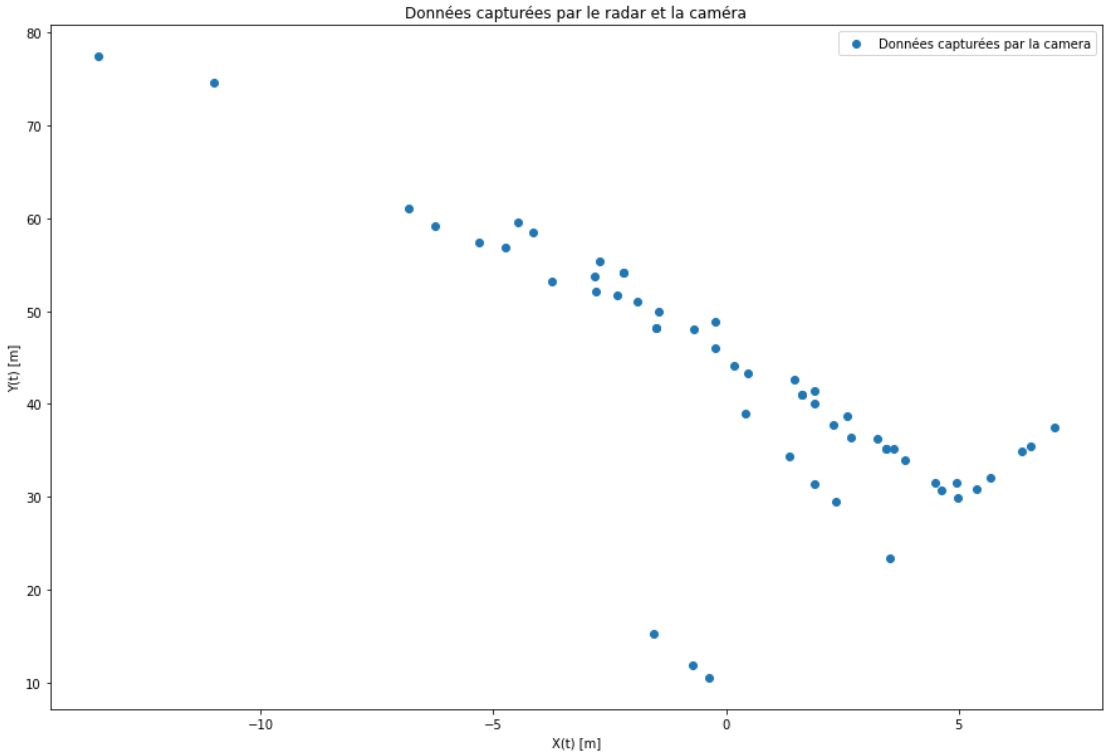


FIGURE 5.6 – Cette Figure illustre les résultats obtenus après extraction des données de la caméra.

Sur la Figure ci-dessus, on constate que l'algorithme d'extraction des données de la caméra est plus stable que ce que l'on pourrait croire. En effet, au Chapitre 3, nous avions remarqué que l'estimation de la position d'un véhicule vu en biais n'était pas toujours bonne. Sur la Figure 5.6, on constate que ceci ne semble pas être le cas. Nous n'avons cependant pas de moyen quantitatif de le vérifier étant donné que nous n'avons pas accès aux positions réelles des véhicules. Nous pouvons cependant raisonner de manière qualitative. En effet, sur la Figure 5.5, on constate

que les véhicules vont apparaître à droite de l'image et disparaître du champ de vue de la caméra à gauche de l'image. Sur base du repère illustré à la Figure 1.10a, il en résulte que les véhicules les plus proches de la caméra auront une position $X(t)$ positive et une position $Y(t)$ de l'ordre d'une vingtaine de mètres. Les véhicules les plus éloignés auront, quant à eux, une position $X(t)$ négative et une position $Y(t)$ de l'ordre de la cinquantaine de mètres.

La stabilité de l'algorithme d'extraction est dûe au fait que, contrairement à la simulation, nous n'avons pas accès aux dimensions réelles des véhicules. Pour pallier ce problème, il a été décidé d'approximer la longueur moyenne d'une voiture par 4.5 [m] et sa largeur par 1.8 [m]. Par ailleurs, sur la Figure 5.6, nous pouvons observer la présence d'outliers, comme ceux situés aux alentours de la position $(X(t), Y(y)) = (-15, 75)$. Ils résultent du fait que les véhicules correspondant à ces positions commencent à sortir du champ de vision de la caméra. Ceci est illustré à la Figure suivante :

FIGURE 5.7 – Illustration d'une voiture commençant à sortir du champ de vision de la caméra

Finalement, le tableau suivant fournit les résultats obtenus tant par la caméra que le radar dans un cas réel illustré à la Figure 5.8.

FIGURE 5.8 – Exemple de véhicules capturés par le système radar-caméra. On ne s'intéresse qu'au véhicule le plus proche du système bimodal.

Données de la caméra				Données du radar			
d [m]	θ [°]	ϕ [°]	v [m/s]	d [m]	θ [°]	ϕ [°]	v [m/s]
38.8103	92.2313	3.85	13.3229	34.3721	88.8766	-11.1606	10.2529

TABLE 5.3 – Résultats obtenus par chaque senseur pour le véhicule le plus proche illustré à la Figure 5.8

Ainsi, on constate que la distance semble correctement extraite par la caméra étant donné qu'elle est proche de celle extraite par le radar. On remarque que les angles ne sont pas bien estimés par le radar, résultant de la mauvaise synchronisation entre les deux senseurs. De plus, grâce à la stabilité de l'algorithme d'extraction des données, on constate que la vitesse estimée par la caméra est fiable.

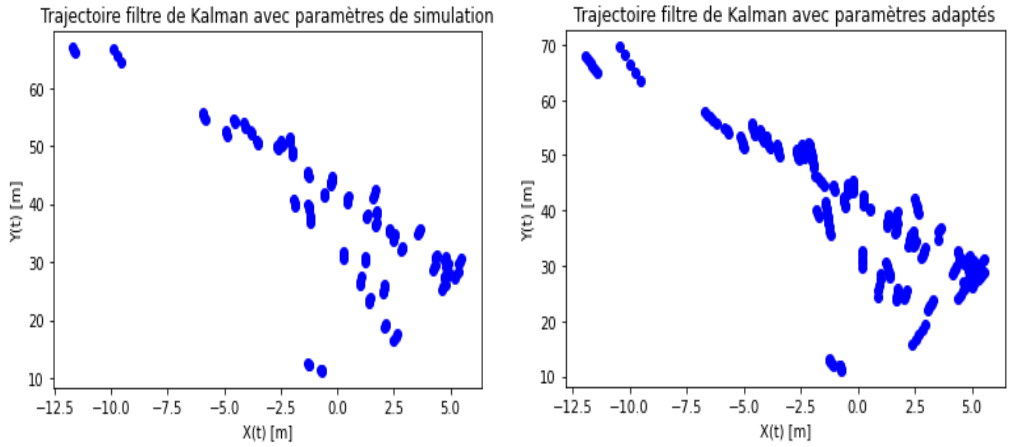
5.4 Résultats

Dans cette section, nous allons comparer les valeurs filtrées renvoyées par nos trois architectures : le filtre de Kalman, le MLP et le Transformer. N'ayant pas accès à la trajectoire réelle parcourue par les véhicules, l'évaluation des mesures se

fera de manière qualitative à l'aide de l'analyse des trajectoires des véhicules. De plus, comme dit précédemment, la trajectoire d'un véhicule ne sera modélisée que par quelques instants temporels. Pour pallier ce problème, nous allons considérer les trajectoires parcourues par plusieurs véhicules et supposer qu'elles sont identiques. Ceci permettant uniquement une meilleure visualisation des résultats. En pratique, le système bimodal observe une route à deux bandes et donc, les trajectoires ne seront pas identiques. Les architectures neuronales du type MLP et transformer ont été entraînées sur les données simulées.

5.4.1 Résultats du filtre de Kalman

La première chose à remarquer est que l'intervalle de temps entre deux échantillons n'est pas constant. Dès lors, les valeurs des paramètres optimisés au Chapitre 4 ne sont plus valables. De plus, à la Figure 5.4, nous avons pu remarquer que la levée des ambiguïtés par le radar ne se faisait pas toujours correctement. Dès lors, on peut considérer que la covariance pour les angles θ et ϕ de la matrice de covariance de mesures R_r n'est plus correcte. Cette même observation peut être faite pour la covariance de la distance d_{cam} ou de la vitesse v_{cam}. On remarque donc un des désavantages du filtre de Kalman. Pour que celui-ci puisse fonctionner correctement, ces matrices doivent être correctement estimées. Cependant, les simulations du carrefour fournies par François-Xavier Inglese ne permettent pas pour l'instant de déterminer des valeurs correctes pour l'estimation de ces matrices. En effet, un certain manque de diversité au niveau des trajectoires des véhicules se fait ressentir.



(a) Trajectoire prédite par le filtre de Kalman avec les valeurs paramètres trouvées lors de la simulation

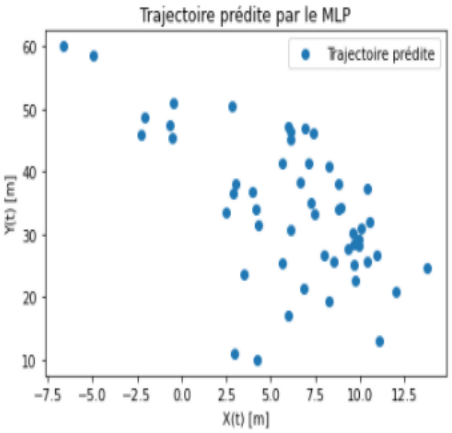
(b) Trajectoire prédite par le filtre de Kalman avec des paramètres adaptés

FIGURE 5.9 – Comparaison des trajectoires avec des paramètres choisis différents

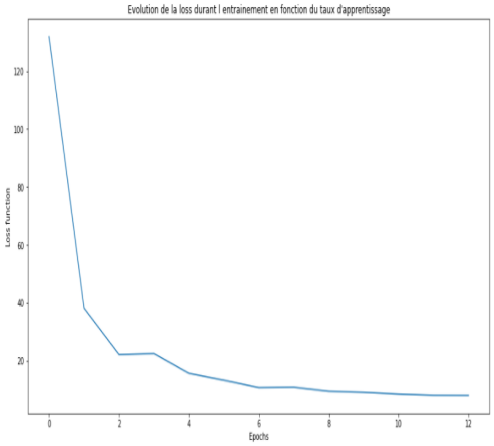
Ceci est confirmé par les Figures 5.9a et 5.9b. On observe que la trajectoire empruntée par les véhicules semble plus visible sur la seconde figure. Il faut cependant noter que, comme expliqué précédemment, contrairement aux indications des fiches techniques, le radar ne fournit pas des données à intervalle constant ce qui rend l'association des détections aux pistes difficile. En effet, les pistes disparaissent avant qu'une détection succincte ne puisse être associée à la piste correspondante.

5.4.2 Résultats du MLP

Dans un premier temps, nous avons entraîné un MLP en lui donnant des données à 8 dimensions : la position et la vitesse des cibles extraites par le radar et la caméra. Le réseau est composé de 128 neurones et nous utilisons des batchs de taille 4. De plus, un taux d'apprentissage $\alpha = 10^{-3}$ est utilisé avec la méthode d'Adam. Comme nous pouvons le remarquer sur la Figure suivante, le réseau est un peu sur-entraîné. Il semble également que le taux d'apprentissage soit un peu trop grand car la fonction coût ne décroît pas de façon monotone.



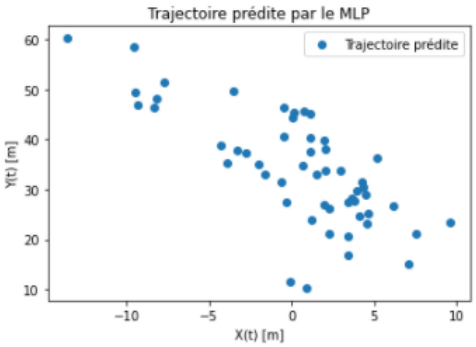
(a) Trajectoire prédite par le MLP



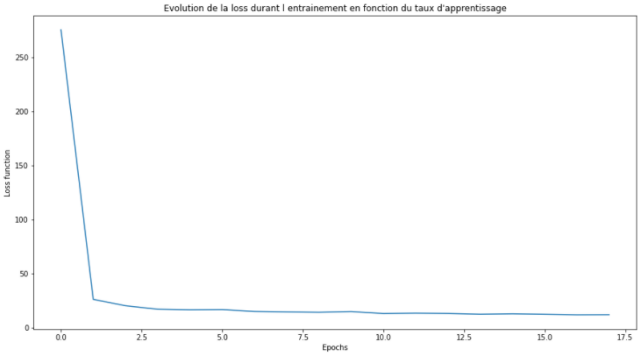
(b) Evolution de la fonction coût au cours de l'entraînement

FIGURE 5.10 – On constate que le MLP est un peu sur-entraîné car la plupart des positions estimées se trouvent en $X(t)$ positif.

Sur la Figure ci-dessus, on constate que le réseau est sur-entraîné. En effet, les positions des véhicules semblent se concentrer du côté des $X(t)$ positifs. En pratique, grâce à l'angle de vue du système radar-caméra, les trajectoires des véhicules devraient évoluer des $X(t)$ positifs vers les $X(t)$ négatifs. De plus, le taux d'apprentissage semble trop grand, ce qui peut mener à un mauvais entraînement du réseau. En prenant un taux d'apprentissage plus petit ($\alpha = 10^{-4}$), nous obtenons de meilleurs résulats :



(a) Trajectoire prédite par le MLP

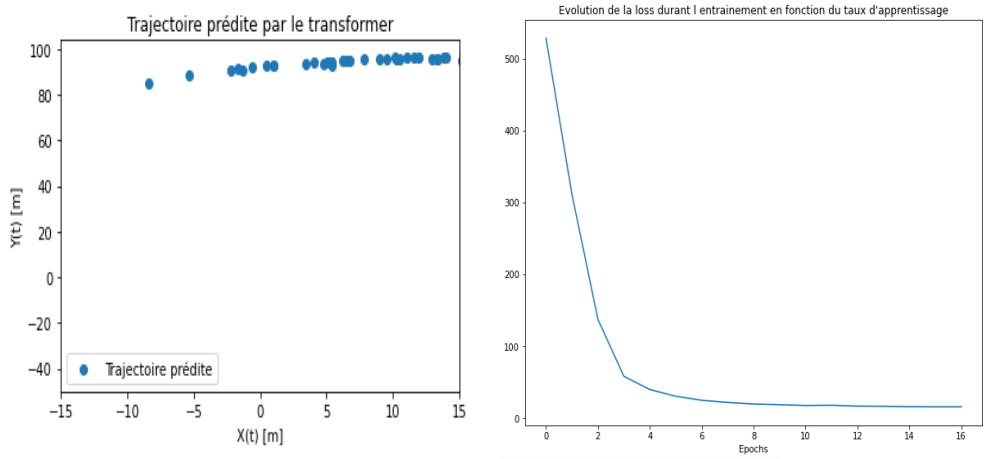


(b) Evolution de la fonction coût au cours de l'entraînement

FIGURE 5.11 – On constate que le MLP est un peu mieux entraîné.

5.4.3 Résultats du Transformer

Dans cette section, nous considérons un transformer à 7 couches, avec 64 neurones sur la couche intermédiaire du MLP. Il est entraîné avec des batchs de taille 16. Nous constatons que le réseau est complètement sur-entraîné et qu'il n'a donc rien appris.

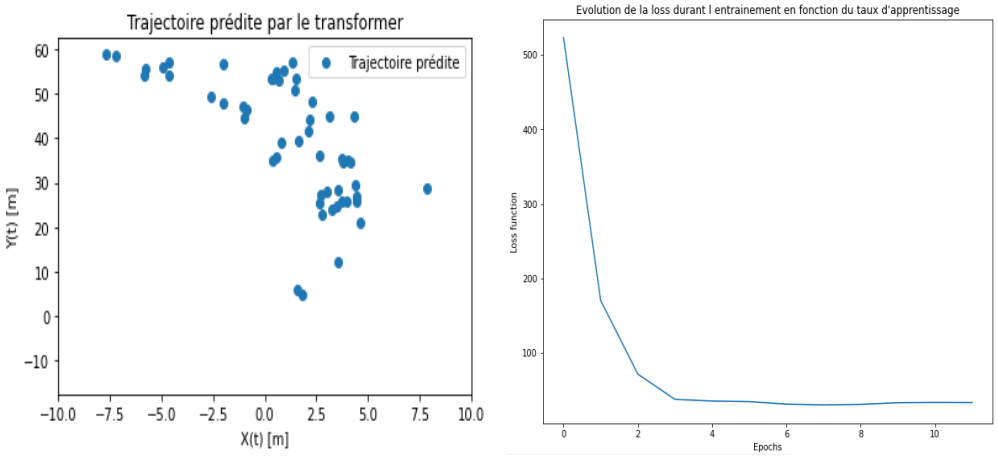


(a) Trajectoire prédite par le transformer

(b) Evolution de la fonction coût au cours de l'entraînement

FIGURE 5.12 – On constate que le transformer est complètement sur-entraîné.

Sur la Figure ci-dessus, on constate que le transformer est complètement sur-entraîné : il n'est pas capable de généraliser. Pour pallier ce problème, il a été décidé de fournir une dimension supplémentaire au réseau : l'aire de la bounding box du véhicule. Nous obtenons les résultats suivants :



(a) Trajectoire prédite par le transformer

(b) Evolution de la fonction coût au cours de l'entraînement

FIGURE 5.13 – On constate que le transformer est un peu mieux entraîné.

On constate donc que le transformer fournit de meilleurs résultats lorsqu'il a une dimension supplémentaire en entrée. Ceci a également été testé avec le MLP mais cela ne change pas fondamentalement les résultats.

5.5 Exemple d'application du système bimodal

Le système bimodal développé dans le cadre de ce travail peut être utilisé dans bon nombre d'applications. Par exemple dans une étude réalisant des statistiques sur un carrefour tel que celui étudié. Ainsi, le système pourrait fournir des statistiques telles que le nombre de voitures, camions et usagers faibles circulant au cours d'une journée. Cette étude est particulièrement intéressante car elle pourrait mener à un développement de la voirie tel que l'ajout de pistes cyclables et de trottoirs. En outre, le système est capable de mesurer la vitesse des véhicules. Dès lors, une étude statistique du carrefour permettrait également de déterminer le profil de vitesse des automobilistes.

Chapitre 6

Améliorations

6.1 Augmentation de la taille du vecteur d'entrée

Dans le cadre de ce travail, le vecteur d'entrée dans les réseaux neuronaux comportait 8 dimensions : la position et la vitesse de chaque véhicule telles que perçues par les senseurs. Cependant, la caméra est capable de fournir beaucoup plus d'information, certaines ayant été abordées brièvement dans le mémoire telles que l'évolution de la bounding box au cours du temps, ou encore l'intervalle de temps entre deux instants d'échantillonnage. De même, le radar comporte des informations supplémentaires non exploitées, principalement grâce à l'effet micro-Doppler, telles que la distance inter-essieux mesurable grâce à la distance séparant le spectre micro-Doppler des roues avant et arrière d'un même véhicule.

L'ajout de ces dimensions supplémentaires permettrait une amélioration de la prédiction fournie par les réseaux. En effet, au plus un réseau de neurone recevra d'information, au plus il sera capable de fournir des prédictions précises. Il faudra toutefois veiller à ce que le modèle ne devienne pas trop gourmand en mémoire vive.

6.2 Passage à des bounding box 3D

Dans le cadre de ce travail, il a été décidé d'utiliser YOLOV5 pour la détection d'objet sur une image. Cet algorithme fournit de bons résultats mais uniquement des bounding boxs 2D. Cette restriction à deux dimensions est une source d'erreur importante pour l'évaluation de la distance radiale d'un véhicule à cause de la méthode développée pour extraire les données de la caméra. En passant à une

bounding box en 3D, la méthode d'estimation de la distance deviendrait plus robuste, car il serait possible de détecter les différentes faces du véhicule et donc d'avoir une meilleure approximation du rapport pixel-longueur effective du véhicule.

6.3 Déploiement en temps réel

D'un coté, le travail proposé repose sur la récolte de données et de l'autre, le traitement de celui-ci à un moment ultérieur. Il serait donc possible d'implémenter ce système en temps réel afin de ne récolter que les informations nécessaires en supprimant les données radar et caméra après extraction des données utiles. Il est possible d'avoir un produit fini fonctionnant en temps réel capable d'enregistrer des évènements lorsque certaines conditions sont remplies telles que des voitures roulant à des vitesses trop élevées ou des situations où des véhicules sont trop proches les un des autres, tout cela dans le but d'augmenter la sécurité routière.

6.4 Synchroniser le radar et la caméra

Dans la validation expérimentale le sujet a été abordé. Le radar renvoie moins de frames que planifié par la fiche technique. Dés lors, une partie des données caméra récoltée est non utilisé, car il nous faut une concordance entre le nombre de véhicules détectées par le radar et la caméra pour que notre algorithme puisse fonctionner.

6.5 Méthode d'association

La méthode d'association utilisée est la minimisation de la différence de distances entre celles relevées par la caméra et le radar. Cette méthode n'est pas très robuste quand de multiples détections sont prises proches les unes des autres. Une solution serait de développer une fonction permettant de prendre les angles des deux senseurs en compte tout en tenant compte des ambiguïtés afin d'obtenir une méthode d'association robuste

Chapitre 7

Conclusion

Tout au long de ce travail, une méthode a été développée pour créer un algorithme robuste et efficace capable de fusionner deux senseurs différents ensemble afin d'analyser la trajectoire de cibles en mouvement sur un lieu donné. Tout d'abord, un simulateur de radar Doppler a été créé sur base de données annotées fournies par le moteur de jeu UnrealEngine4. Ce simulateur utilise les données de la situation présentes sur la caméra afin d'en produire l'équivalent sur le radar, la simulation prenant en compte un effet particulier observable sur les véhicules, l'effet micro-Doppler.

Ensuite un algorithme de reconnaissance d'objets YOLOv5 a été choisi pour identifier les véhicules sur les images caméra tandis qu'un algorithme d'identification a été développé pour identifier des objets sur les données radar. On remarquera qu'une limitation de la simulation caméra fut trouvée dans le fait que les images caméra soient de synthèse sans distance focale connue et sans flou. Une alternative a été trouvée en utilisant le champ de vision de ce senseur, bien que donnant des résultats moins précis. Notre méthode d'association données radar-caméra se basant sur une minimisation de la différence de distance quadratique, celle-ci reste un point faible de notre travail, du fait de l'augmentation du risque d'erreur lorsque plusieurs véhicules sont proches l'un de l'autre. L'association effectuée, il est possible d'enlever les ambiguïtés des angles du radar grâce aux angles fournis par la caméra.

L'extraction terminée, les performances des trois différentes architectures implémentées ont pu être comparées. Nous avons observé que les résultats du Transformer ,bien que moins bons que ceux du MLP en terme de MSE et nécessitant un temps d'exécution plus important, produisent une trajectoire plus lisse semblable au filtre de Kalman. Cela montre tout le potentiel de l'architecture. Cependant, beaucoup d'information n'a pas été utilisée dans son implémentation, comme par exemple le

temps entre deux échantillons. Dès lors, il semble fort possible d'améliorer les performances du Transformer, bien que cela risque de demander un temps d'exécution plus important. Cette architecture présente donc un potentiel innovant en matière de fusion de données multimodales et suivi de cibles comparée une architecture linéaire simple fort dépendante de ses paramètres et d'une architecture non-linéaire fonctionnelle mais simple d'esprit.

Enfin, nos algorithmes entraînés sur des mesures simulées ont été testés sur un cas réel afin de valider les simulations. En extrayant les données, nous avons pu remarquer que le taux d'acquisition des données radars étaient beaucoup plus faible que prévu, rendant la vérification des algorithmes plus difficile. N'ayant en plus pas les mesures précises des véhicules à un instant t, une vérification visuelle des trajectoires prédites a été faite pour montrer que les trajectoires correspondent bien aux pistes routières empruntées par les cibles.

Pour rendre ce projet apte à fonctionner en temps réel, il reste cependant encore quelques points à améliorer, notamment l'acquisition des données à des intervalles régulier par le radar ou encore l'entraînement des réseaux de neurones sur une plus grande variété de données afin de le rendre plus robuste.

Chapitre 8

Annexe

A Vitesses maximale et minimale d'une roue captée par le radar

Dans cette annexe, nous développons l'expression analytique des vitesses maximale et minimale d'une roue telles que captées par le radar. Il en résulte un étalement de vitesse sur les heatmaps distance/vitesse. Le raisonnement suivant se base sur [4].

A.1 Cinématique d'une roue d'un véhicule.

Une roue est modélisée par un cercle de rayon R et de centre C dont le déplacement d'un point M peut être décrit par les deux opérations géométriques suivantes :

— une translation de vitesse v ;

— une rotation autour du point C d'un angle $\alpha(t)$ telle que $\alpha(t) = \frac{vt}{R} = \Omega t$ où Ω est la vitesse angulaire de la roue.

Ceci est schématisé sur la Figure suivante.

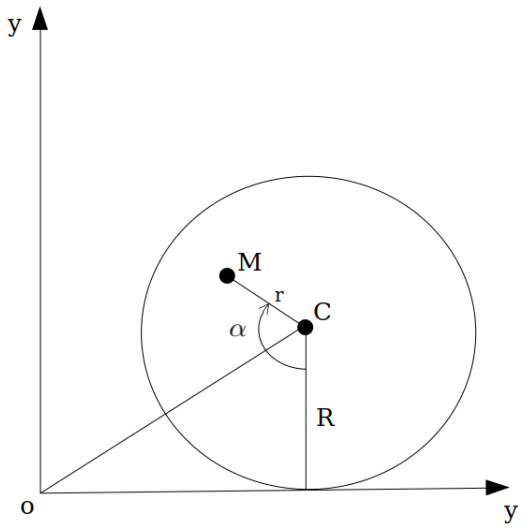


FIGURE 8.1 – Modèle de la roue utilisée pour l'analyse de la cinématique

Par la suite, on suppose l'hypothèse de roulement sans glissement satisfaite, c'est-à-dire que la vitesse de la roue au point de contact au sol est nulle. On peut alors décrire la trajectoire du point M par

$$O\vec{M}(t) = \begin{cases} X_M(t) = R\alpha - r\sin(\alpha) \\ Y_M(t) = R - r\cos(\alpha) \\ Z_M(t) = 0 \end{cases} \tag{8.1}$$

où r correspond à la distance entre le centre de la roue et le point M ($0 \leq r \leq R$).

A.2 Vitesse des points de la roue

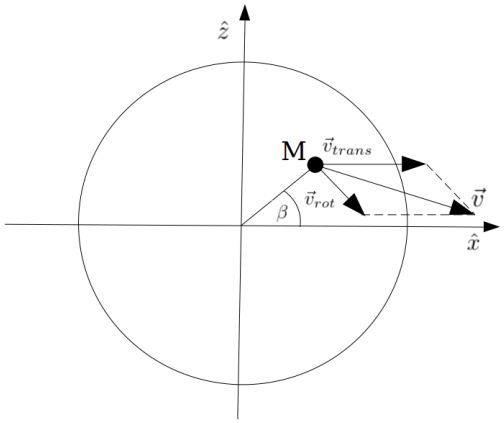


FIGURE 8.2 – Repère R1 utilisé

Considérons maintenant un repère R1 $(O, \hat{x}, \hat{y}, \hat{z})$, tel qu'illustré à la Figure 8.2, centré sur l'axe de la roue se déplaçant à une vitesse $\vec{v}_{abs}$. Celle-ci pouvant se décomposer en une vitesse de translation $\vec{v}_{trans}$ et une vitesse de rotation $\vec{v}_{rot}$, on a :

$$\vec{v}(x, z) = \vec{v}_{trans} + \vec{v}_{rot} \tag{8.2}$$

Les coordonnées du point M s'exprimant comme :

$$M(x, y, z) = \begin{cases} x = r\cos(\beta) \\ y = 0 \\ z = r\sin(\beta) \end{cases} \tag{8.3}$$

On peut définir les vitesses de translation et de rotation de la façon suivante :

$$\vec{v}_{trans} = v_{abs}\hat{x} \tag{8.4}$$

$$\vec{v}_{rot} = \frac{v_{abs}}{R}(z\hat{x} - x\hat{z}) \tag{8.5}$$

La norme du vecteur vitesse vaut donc :

$$||\vec{v}(x, y, z)|| = \frac{v_{abs}}{R}\sqrt{(R + z)^2 + x^2} \tag{8.6}$$

On peut remarquer qu'à la limite du cercle, c'est-à-dire en $x^2 + z^2 = R^2$, la condition de roulement sans glissement est satisfaite. En effet $||\vec{v}(0, y, -R)|| = \frac{v_{abs}}{R}\sqrt{2R(R+z)}\,|_{z=-R} = 0$

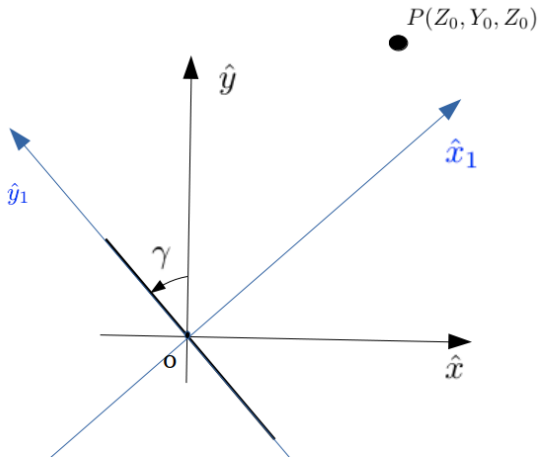


FIGURE 8.3 – Repère R2 utilisé

La vitesse, telle que définie par l'Equation (8.2), correspond à la vitesse en tout point de la roue. Cependant, il ne s'agit pas de la vitesse perçue par le radar. En effet, cette dernière correspond à la vitesse radiale définie comme la projection orthogonale du vecteur vitesse sur le vecteur de visée correspondant à la droite reliant un observateur P placé en (X_0, Y_0, Z_0) dans un repère R2 de l'espace à un point M de la roue. Ce repère R2 $(O, \hat{x}_1, \hat{y}_1, \hat{z}_1)$, illustré à la Figure 8.3, a comme origine le centre de la roue et l'axe OZ est parallèle à celui du repère R1. Ainsi, pour passer du repère R1 au repère R2, il faut effectuer une rotation autour de l'axe OZ d'un angle γ ($0 \leq \gamma \leq \pi$) :

$$R_{12} = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{8.7}$$

On peut donc réécrire le vecteur $\overrightarrow{OM}$ et le vecteur vitesse $\vec{v}$ dans le nouveau repère R2 :

$$\overrightarrow{OM} = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ 0 \\ z \end{pmatrix} = \begin{pmatrix} x\cos\gamma \\ x\sin\gamma \\ z \end{pmatrix} \tag{8.8}$$

$$\vec{v} = \frac{v_{abs}}{R} \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R+z \\ 0 \\ -x \end{pmatrix} = \begin{pmatrix} (R+z)\cos\gamma \\ (R+z)\sin\gamma \\ -x \end{pmatrix} \tag{8.9}$$

La vitesse radiale v_{rad} étant définie comme la projection du vecteur vitesse sur le vecteur de visée, on a :

$$v_{rad} = \frac{\vec{v} \cdot \vec{PM}}{||\vec{PM}||} \tag{8.10}$$

Le vecteur de visée $\overrightarrow{PM}$ s'exprime comme :

$$\overrightarrow{PM} = -\overrightarrow{OP} + \overrightarrow{OM} = \begin{cases} X_{PM} = -X_0 + x\cos(\gamma) \\ Y_{PM} = -Y_0 + x\sin(\gamma) \\ Z_{PM} = -Z_0 + z \end{cases} \tag{8.11}$$

Les dimensions de la roue étant négligeables par rapport à la distance entre l'observateur et la roue, on a $\vec{PO} \gg \vec{OM}$. Donc, le vecteur $\overrightarrow{PM}$ devient :

$$\vec{PM} = \begin{cases} X_{PM} = -X_0 \\ Y_{PM} = -Y_0 \\ Z_{PM} = -Z_0 \end{cases} \tag{8.12}$$

On en déduit donc que sa norme vaut

$$||\vec{PM}|| = \sqrt{X_0^2 + Y_0^2 + Z_0^2} = D \tag{8.13}$$

Par conséquent, l'Equation (8.10) nous donne :

$$v_{rad}(x, y, z) = -\frac{v_{abs}}{RD}(X_0(R+z)\cos(\gamma) + Y_0(R+z)\sin(\gamma) - Z_0 x) \tag{8.14}$$

On pose :

$$\frac{X_0}{D} = \sin(\theta)\cos(\phi) \tag{8.15}$$

$$\frac{Y_0}{D} = \sin(\theta)\sin(\phi) \tag{8.16}$$

$$\frac{Z_0}{D} = \cos(\theta) \tag{8.17}$$

L'Equation (8.14) devient donc finalement :

$$v_{rad}(x,y,z) = -\frac{v_{abs}}{R}\left((R+z)\sin\theta\left(\cos\phi\cos\gamma+\sin\phi\sin\gamma\right)-x\cos\theta\right) \tag{8.18}$$

$$= -\frac{v_{abs}}{R}\left((R+z)\sin(\theta)\cos\left(\phi-\gamma\right)-x\cos(\theta)\right) \tag{8.19}$$

Cette dernière équation exprime la vitesse perçue par la radar à chaque point appartenant à la roue. Par la suite, nous allons déterminer ses valeurs maximale et minimale

A.3 Vitesses maximale et minimale

L'Equation (8.19) permet de calculer la vitesse radiale de n'importe quel point M de la roue. Intéressons-nous maintenant aux vitesses les plus extrêmes que l'observateur est susceptible de percevoir, c'est-à-dire aux vitesses radiales maximale et minimale. Comme nous sommes en condition de roulement sans glissement, ces valeurs extrêmes se trouveront sur la circonférence de la roue et le point M aura comme coordonnées $M(R\cos\beta, 0, R\sin\beta)$. En remplaçant dans l'Equation (8.19) les coordonnées x et z par ces coordonnées, on constate que la vitesse radiale est indépendante du rayon de la roue. En effet, on a :

$$v_{rad}(\beta) = -v_{abs}\left((1+\sin\beta)\sin\theta\cos\left(\phi-\gamma\right)-\cos\beta\cos\theta\right) \tag{8.20}$$

Les valeurs extrêmes recherchées correspondant aux valeurs pour lesquelles la dérivée de la vitesse radiale s'annule, on a :

$$\frac{dv_{rad}}{d\beta} = -v_{abs}\left(\cos\beta\sin\theta\cos\left(\phi-\gamma\right)+\sin\beta\cos\theta\right) \tag{8.21}$$

$$= 0 \tag{8.22}$$

Donc, la valeur de β annulant cette équation est telle que :

$$\tan(\beta) = \tan\theta cos(\phi-\gamma) \tag{8.23}$$

Autrement dit,

$$\beta_1 = \arctan(-\tan\theta\cos(\phi-\gamma)) \tag{8.24}$$

$$\beta_2 = \arctan(-\tan\theta\cos(\phi-\gamma))+\pi \tag{8.25}$$

On en déduit finalement les valeurs minimale et maximale de la vitesse radiale :

$$\begin{cases} v_{min} = -v_{abs}\left((1-\sin\beta)\sin\theta\cos(\phi-\gamma)+\cos\beta\cos\theta\right) \\ v_{max} = -v_{abs}\left((1+\sin\beta)\sin\theta\cos(\phi-\gamma)-\cos\beta\cos\theta\right) \end{cases} \tag{8.26}$$

En posant $u = \arctan(\tan\theta\cos(\phi-\gamma))$ et en utilisant les relations trigonométriques suivantes :

$$\begin{cases} \sin(\arctan(u)) = \dfrac{u}{\sqrt{1+u^2}} \\ \cos(\arctan(u)) = \dfrac{1}{\sqrt{1+u^2}} \end{cases} \tag{8.27}$$

L'Equation (8.26) peut s'exprimer comme :

$$v_{min} = -v_{abs}\left((1-\frac{u}{\sqrt{1+u^2}})\sin\theta\cos(\phi-\gamma) + \frac{1}{\sqrt{1+u^2}}\cos\theta\right) \tag{8.28}$$

$$v_{max} = -v_{abs}\left((1+\frac{u}{\sqrt{1+u^2}})\sin\theta\cos(\phi-\gamma) - \frac{1}{\sqrt{1+u^2}}\cos\theta\right) \tag{8.29}$$

Références

[1] Slootmans Freya. *Rapport Statistique 2020 – Accidents de la route 2019*. https://www.vias.be/publications/Statistisch%20rapport%202020%20-%20verkeersongevallen%202019/Rapport_statistique_2020_Accidents_de_la_route_2019.pdf. (Visité le 08/06/2021).

[2] Gauthier Rotsart de Hertaing Tanguy Dehon Thomas Delsart. *LFSAB1508 : Projet 4 Electricité Radar Doppler*. 2019.

[3] Thomas Heuschling Morgan Leclerc Kevin De Sousa Thibaud Hauptmann. *Rapport de projet - Radar numérique*. 2019.

[4] Antoine Ghaleb. « Analyse des micro-Doppler de cibles mobiles déformables en imagerie radar. Traitement du signal et de l'image [eess.SP] ». Thèse de doct. Télécom ParisTech, 2009.

[5] Younis Marwin. *Chapter 2 Antenna Arrays and Beam-Forming*. Déc. 2020.

[6] Inductiveload. https://commons.wikimedia.org/wiki/File:Spherical_Coordinates_(Colatitude,_Longitude)_(b).svg. (Visité le 11/06/2021).

[7] Alexis Duflot. *Conception d'un senseur intégré multimodal pour l'observation des routes*. 2020, p. 67-86.

[8] Moacir Antonelli Ponti et al. « Everything you wanted to know about deep learning for computer vision but were afraid to ask ». In : *2017 30th SIBGRAPI conference on graphics, patterns and images tutorials (SIBGRAPI-T)*. IEEE. 2017, p. 17-41.

[9] Alexey Bochkovskiy, Chien-Yao Wang et Hong-Yuan Mark Liao. *YOLOv4 : Optimal Speed and Accuracy of Object Detection*. 2020. arXiv : 2004.10934 [cs.CV].

[10] Glenn Jocher et al. *ultralytics/yolov5 : v5.0 - YOLOv5-P6 1280 models, AWS, Supervise.ly and YouTube integrations*. Version v5.0. Avr. 2021. doi : 10.5281/zenodo.4679653. url : https://doi.org/10.5281/zenodo.4679653.

[11] Matthijs Hollemans. *Real-time object detection with YOLO*. https://machinethink.net/blog/object-detection-with-yolo/. (Visité le 28/05/2021).

[12] Tsung-Yi Lin et al. *Microsoft COCO : Common Objects in Context*. 2015. arXiv : 1405.0312 [cs.CV].

[13] Jessada Phattaralerphong et Hervé Sinoquet. « A method for 3D reconstruction of tree crown volume from photographs : assessment with 3D-digitized plants ». In : *Tree Physiology* 25.10 (2005), p. 1229-1242.

[14] Christophe De Vleeschouwer et Laurent Jacques. *LELEC2885 : Image processing and computer vision [Geometry4Vision]*. 2020, p. 1-13, 40.

[15] Wikipédia. *Centre d'inertie*. fr.wikipedia.org/wiki/Centre_d%27inertie. (Visité le 16/04/2021).

[16] Sébastien Cardinael et Arthur Malcourant. *Conception d'un senseur intégré multimodal pour l'observation des routes*. 2018, p. 14-16.

[17] Tod Musgrave. *Focal Lengths and Angle-of-View (AOV) explained*. https://www.svconline.com/the-wire/focal-lengths-and-angle-of-view-aov-explained. (Visité le 10/05/2021).

[18] Jeff Atwood. *Widescreen and FOV*. https://blog.codinghorror.com/widescreen-and-fov/. (Visité le 10/05/2021).

[19] *data sheet K-MD2*. http://rfbeam.com/files/products/21/downloads/Datasheet_K-MD2.pdf.

[20] Q. Gan et C.J. Harris. « Comparison of two measurement fusion methods for Kalman-filter-based multisensor data fusion ». In : *IEEE Transactions on Aerospace and Electronic Systems* 37.1 (2001), p. 273-279. doi : 10.1109/7.913685.

[21] Sebin Park et al. « Measurement Noise Recommendation for Efficient Kalman Filtering over a Large Amount of Sensor Data ». In : *Sensors* 19.5 (2019). issn : 1424-8220. doi : 10.3390/s19051168. url : https://www.mdpi.com/1424-8220/19/5/1168.

[22] Awhan Mohanty. *Multi layer Perceptron (MLP) : Models on Real World Banking Data*. https://becominghuman.ai/multi-layer-perceptron-mlp-models-on-real-world-banking-data-f6dd3d7e998f. (Visité le 02/05/2021).

[23] Sagar Sharma. *What the Hell is Perceptron*. https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53. (Visité le 10/05/2021).

[24] Michel Verleysen et John Lee. *LELEC2870 : Machine learning : regression and dimensionality reduction [Nonlinear regression with Multi-Layer Perceptrons]*. 2020, p. 1-38.

[25] Diederik P Kingma et Jimmy Ba. « Adam : A method for stochastic optimization ». In : *arXiv preprint arXiv :1412.6980* (2014).

[26] Francesco Giuliari et al. « Transformer networks for trajectory forecasting ». In : *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE. 2021, p. 10335-10342.

[27] Yicheng Liu et al. « Multimodal Motion Prediction with Stacked Transformers ». In : *arXiv preprint arXiv :2103.11624* (2021).

[28] Jean-Benoit Delbrouck et al. « A Transformer-based joint-encoding for Emotion Recognition and Sentiment Analysis ». In : *arXiv preprint arXiv :2006.15955* (2020).

[29] Ashish Vaswani et al. « Attention is all you need ». In : *arXiv preprint arXiv :1706.03762* (2017).

[30] Kaiming He et al. « Deep residual learning for image recognition ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, p. 770-778.

[31] Ketan Doshi. *Transformers Explained Visually (Part2) : How it works, step-by-step*. `https://towardsdatascience.com/transformers-explained-visually-part-2-how-it-works-step-by-step-b49fa4a64f34`. (Visité le 13/05/2021).

[32] Jimmy Lei Ba, Jamie Ryan Kiros et Geoffrey E Hinton. « Layer normalization ». In : *arXiv preprint arXiv :1607.06450* (2016).

[33] Wikipédia. *Camera calibration*. `en.wikipedia.org/wiki/Camera_calibration`. (Visité le 28/04/2021).

[34] Ivan Orsolic. *Camera calibration : Explaining camera distorsions*. `https://ori.codes/artificial-intelligence/camera-calibration/camera-distortions/`. (Visité le 28/04/2021).

[35] Thomas Schon. *Dynamic Vision[Lecture2 : Camera models and Calibration]*. `https://www.control.isy.liu.se/student/graduate/DynVis/Lectures/le2.pdf`. (Visité le 28/04/2021).

[36] Dan Carr. *How to calculate field of view in photography*. `https://shuttermuse.com/calculate-field-of-view-camera-lens/#:~:text=Equation%20For%20Calculating%20Angle%20of%20View%20Simple%20trigonometry,in%20degrees%2C%20you%20do%20not%20need%20that%20bit%21`. (Visité le 28/04/2021).

[37] Ketan Doshi. *Transformers Explained Visually (Part1) : Overview of Functionality*. https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452. (Visité le 13/05/2021).

[38] Ketan Doshi. *Transformers Explained Visually (Part3) : Multi-Head Attention, deep dive*. https://towardsdatascience.com/transformers-explained-visually-part-3-multi-head-attention-deep-dive-1c1ff1024853. (Visité le 13/05/2021).

[39] Amirhossein Kazemnejad. *Transformer Architecture : The Positional Encoding*. https://kazemnejad.com/blog/transformer_architecture_positional_encoding/. (Visité le 13/05/2021).

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**