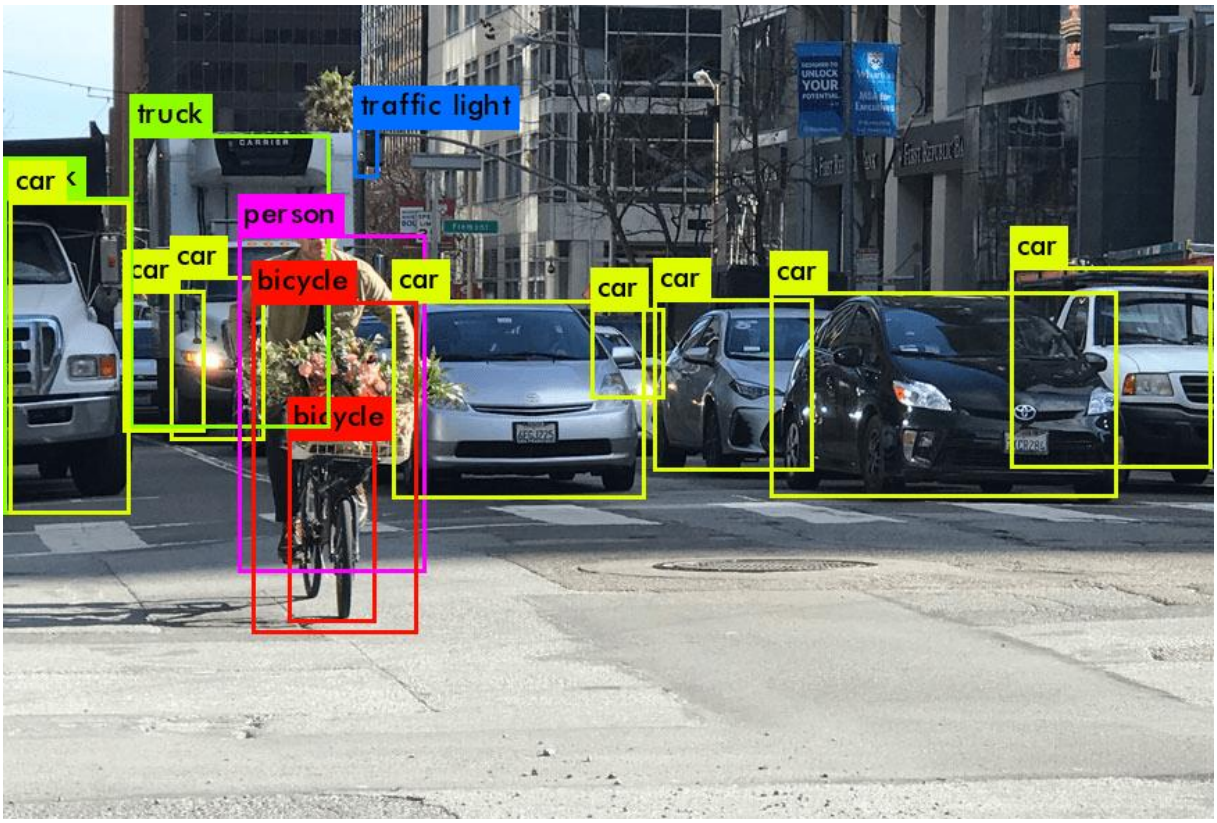
Projet Vision : Détection et suivi 3D de voitures dans des séquences routières



Etudiants : Quentin BERNARD
Lucas ISSARD
Etienne TORZINI

Date : Janvier 2022 / Février 2022

Résumé

L'objectif de notre projet est de réaliser un programme capable de détecter et de suivre des véhicules dans des séquences vidéo issues d'une tête stéréoscopique et de les localiser en 3D par rapport à la voiture qui embarque cette tête. Ce projet a de nombreux champs d'application, notamment pour les véhicules autonomes, là où l'intelligence du véhicule doit être capable de reconnaitre et de comprendre son environnement afin de réagir en conséquence. La détection des autres véhicules mais aussi des piétons est donc nécessaire. Dans notre cas, on se focalisera seulement sur la détection de véhicules. Notre programme sera réalisé en langage Python en s'appuyant notamment sur la bibliothèque OpenCv. Cette bibliothèque est spécialisée dans le traitement d'image en temps réel. Nos programmes seront testés sur une séquence de la base de données KITTI. Cette base de données fournit des séquences vidéo multi-vues dans différents contextes (ville, autoroute…) avec les paramètres de calibrage intrinsèque et extrinsèque des caméras.

Mot clés : Viola & Jones, MedianFlow, Triangulation, détection et suivi de véhicule

Sommaire

Table des figures

Introduction

Les véhicules autonomes et plus généralement les systèmes d'aide à la conduite que l'on retrouve sur la plupart des véhicules modernes ont besoin de reconnaitre parfaitement leur environnement dans le but de circuler en toute sécurité.

De nos jours, plusieurs technologies permettent de détecter des objets dans l'environnement extérieur de la voiture. On peut par exemple penser aux radars embarqués, aux Lidars ou encore aux caméras simples ou stéréoscopiques. Dans notre cas, nous avons à notre disposition une base de données composée d'images de deux caméras stéréoscopiques situées sur le toit d'une voiture accompagnées des données de calibration. Cette base de données est fournie par le projet KITTI. Notre objectif est de créer une application permettant de reconnaître, de suivre et de localiser les autres véhicules dans une séquence vidéo routière.

Dans ce rapport, nous commencerons par une étude bibliographique ainsi que le détail de notre méthode permettant de gérer la détection et le suivi de voitures dans l'environnement. Par la suite, nous présenterons la mise en correspondance des deux images de la tête stéréoscopique afin de déterminer la position dans l'espace des autres véhicules.

1. Détection de Véhicule

A. Etude bibliographique

La reconnaissance d'objets ou de formes est une discipline importante qui a de nombreuses implications dans le domaine de la reconnaissance de caractères manuscrits, de la biométrie ou encore des véhicules autonomes.

La première étape de notre projet est de réussir à détecter de manière fiable les véhicules dans le champ de vision des caméras. La difficulté est de réussir à détecter un maximum de véhicules présents dans le champ de vision tout en évitant les erreurs. Pour cela, plusieurs méthodes existent.

Codage rétinien

La méthode la plus simple est le codage rétinien. Elle consiste à reconnaître une forme spécifique sur une image en la comparant directement à une base de données étiquetée. Bien que très simple à mettre en œuvre et particulièrement efficace pour la reconnaissance de caractères manuscrits, cette méthode ne peut pas être utilisée pour la détection de véhicules. En effet, elle a pour inconvénient de ne pas tolérer les déformations et de devoir centrer la forme à reconnaitre dans une fenêtre de taille fixe. Cette opération est facile à réaliser dans le cadre du traitement d'un texte par exemple, mais est nettement plus compliquée à réaliser pour du traitement de séquences routières où les véhicules à détecter ont des tailles, des couleurs, des formes et des orientations différentes. Il faut donc se tourner vers une méthode plus adaptée.

Intelligence artificielle

Depuis 2012, grâce aux travaux d'Alex Krizhevsky, l'intelligence artificielle et plus précisément les réseaux de neurones convolutifs ont permis des avancées majeures dans le domaine de l'interprétation automatique d'images par ordinateur. Leurs performances sans égal intègrent aujourd'hui bon nombre de système. Le principe de fonctionnement repose sur la reconnaissance d'objets grâce à un réseau de neurones pré-entraîné à l'aide d'une base de données étiquetée. Bien qu'étant très probablement la solution la plus avancée à ce jour dans le domaine de la reconnaissance d'objets, cette technologie nécessite beaucoup de ressources en matière de puissance de calcul. L'intégration d'une intelligence artificielle dans notre projet serait réalisable mais les temps d'exécution seraient probablement élevés, ce qui impliquerait un taux de rafraîchissement assez faible. On se tourne donc vers un dernier algorithme, plus rapide.

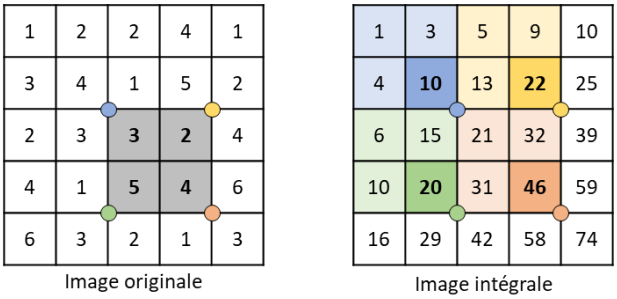
Algorithme de Viola et Jones

L'algorithme de Viola et Jones quant à lui a été présenté en 2001 par les chercheurs Paul Viola et Michael Jones. Il repose sur le principe des filtres de Haar pour la détection et des classifieurs en cascade afin d'accélérer le traitement. Cet algorithme offre de bonnes performances en détection et une faible consommation en ressources. Il pourra donc s'exécuter rapidement comparé à un réseau de neurones convolutif par exemple.

B. Viola et Jones

L'algorithme se présente en 3 principales étapes :

- Calcul de l'image intégrale : L'image intégrale représente l'une des principales innovations apportées par Viola et Jones. Elle est calculée une seule et unique fois par image à traiter. Dans cette image intégrale, chaque pixel contient la valeur de la somme de tous les pixels situés au-dessus et à gauche. Elle peut donc se calculer facilement de manière itérative. Une fois calculée, elle permet d'accélérer grandement les calculs car elle permet d'obtenir la somme des valeurs des pixels d'une zone de l'image par seulement 3 opérations, peu importe la taille de la zone à traiter (*figure 1*).



$$3 + 2 + 5 + 4 \quad = \quad 46 - 20 - 22 + 10 \quad = \quad 14$$

Figure 1 : Image intégrale

- Calcul des caractéristiques de Haar : En s'appuyant sur l'image intégrale calculée précédemment, on calcule les réponses des caractéristiques de Haar. Dans notre projet, par manque de temps, on utilise un modèle de classifieur pré-entraîné stocké au format XML. Ce modèle contient les positions ainsi que les tailles des filtres de Haar à appliquer à l'image afin de détecter des voitures. Grâce à l'image intégrale, les réponses des filtres de Haar peuvent être calculées en un temps record, peu importe leur taille ou leur position dans l'image. Il peut donc y avoir un grand nombre de filtre de Haar à calculer sans que cela impact trop les performances du système.

- Sélection par boosting : Toujours dans l'optique d'améliorer les temps de calcul, plutôt que de calculer tous les filtres de Haar et de conclure sur la présence ou non de véhicules (classifieur fort), on utilise une cascade de classifieurs faibles. A chaque itération, on vérifie si la condition est validée. Si elle l'est, on continue en vérifiant la prochaine condition. Au contraire, si elle ne l'est pas, on rejette immédiatement sans passer par les autres phases de calculs suivantes.

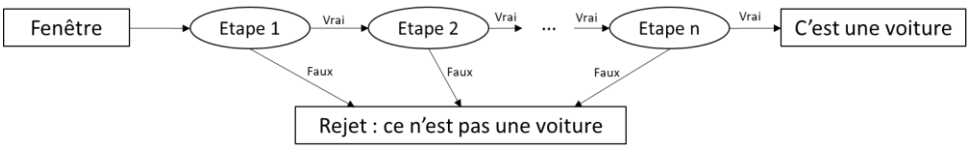


Figure 2 : Sélection par boosting

Cet algorithme est directement intégré dans notre programme à l'aide de la bibliothèque OpenCV qui fournit la fonction `detectMultiScale`. Par soucis de gain de temps, on utilisera un classifieur pré-entraîné pour la détection de voitures, fourni au format XML.

2. Suivi de Véhicule

A. Etude bibliographique

Pour ce qui est du suivi de véhicules, on utilisera aussi les fonctions proposées dans la librairie OpenCV. Il en existe plusieurs avec chacune leurs avantages et leurs inconvénients :

- Boosting : Basé sur un algorithme de machine learning et les filtres de Haar, il est disponible depuis plus de 10 ans. Son fonctionnement est correct mais se révèle plutôt lent et moins fiable que les algorithmes actuels.
- MIL : Offre une meilleure précision que le tracker Boosting mais sa détection des erreurs est médiocre.
- KCF (Kernelized Correlation Filters) : Plus rapide que les algorithmes précédents mais a des difficultés à suivre les objets lorsqu'ils sont en partie occultés.
- CSRT : Offre une grande précision mais est plus lent.
- MedianFlow : Très performant pour le suivi d'objets relativement lents et bonne gestion des erreurs.
- MOSSE : Optimisé purement pour la vitesse. Très rapide mais pas aussi précis que les autres algorithmes.
- GOTURN : Basé sur un algorithme de deep learning. Difficile à entraîner et à utiliser.

Source : https://www.pyimagesearch.com/2018/07/30/opencv-object-tracking/

B. MedianFlow

Compte tenu du cahier des charges de notre application, c'est-à-dire le suivi de véhicules se déplaçant à une vitesse relativement faible sur l'image et sortant régulièrement du champ de vision, il a été choisi d'utiliser l'algorithme MedianFlow qui gère particulièrement bien la sortie du champ de vision des objets suivis.

En réalité, à chaque nouvelle image le traceur compare la trajectoire des points de l'objet à une trajectoire estimée. Si l'écart entre les deux est trop grand, l'algorithme considère qu'il a fait une erreur et cesse de suivre ce point. Ce système permet de gérer de manière particulièrement efficace les occultations ou les sorties de champ des objets.

Dans l'algorithme MedianFlow, ce suivi s'effectue sur une matrice de points placés sur l'objet à suivre. L'algorithme accepte qu'un certain nombre de points ne soient pas reconnus comme dans le cas d'une occultation partielle par exemple (*figure 3*).

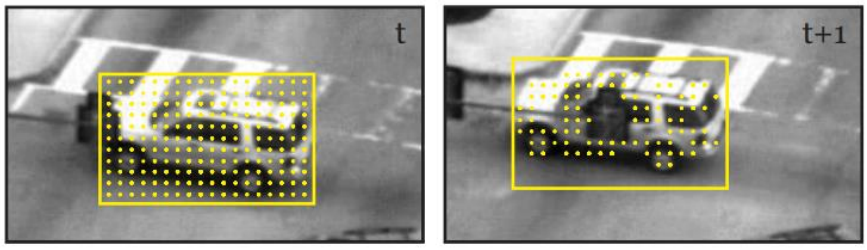


Figure 3 : Occultation partielle d'un véhicule suivi par MedianFlow

Source : Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Forward-backward error: Automatic detection of tracking failures. In Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 2756–2759. IEEE, 2010.*

3. Mise en correspondance et localisation 3D

Une fois la partie détection et suivi des véhicules terminée, on passe à leur localisation dans l'espace. Une seule caméra ne permet pas de voir en 3 dimensions. Ainsi, pour localiser les véhicules, il est nécessaire d'ajouter une ou plusieurs caméras. Le but premier est de mettre en correspondance les points sur les deux images prises en même temps mais avec des caméras situées à des positions légèrement différentes. Ensuite, pour déterminer la position du véhicule, on utilise la triangulation.

A. Mise en correspondance

Dans notre cas, nous avons à notre disposition des photos de 2 caméras cote à cote. Nous allons prendre le cas ou l'on utilise notre algorithme de détection de véhicule sur la photo 1 soit la caméra de gauche (repère R1). Notre algorithme nous retourne la position du véhicule en pixel. La première chose à faire de retrouver cette position sur la photo 2 soit la caméra de droite (Repère R2) afin de localiser le véhicule avec la triangulation.

Pour notre projet, on utilise la base de données KITI comme fichier de test. Cette bibliothèque contient un ensemble d'image et de séquence en multi vues avec les différents paramètres de calibrage des caméras (K1 et K2).

Dans notre cas, le schéma ci-dessous représente notre situation. Il y a deux caméras qui prennent les images en même temps. La première à la position O1 et la deuxième à la position O2.

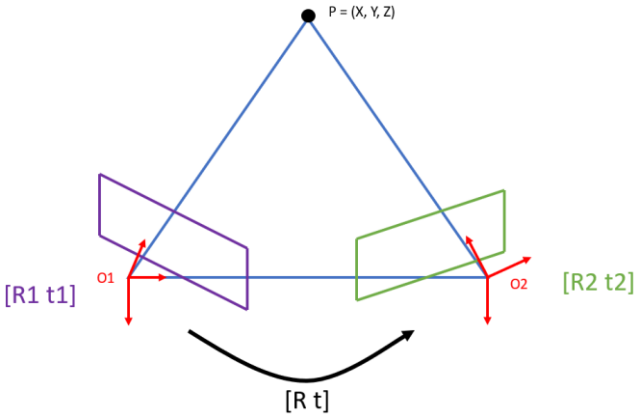


Figure 4 : Schéma des deux caméras

Dans un premier temps il est nécessaire de déterminer la rotation et la translation entre les caméras. Dans la bibliothèque, on a à notre disposition les rotations et les translations entre les caméras et un repère.

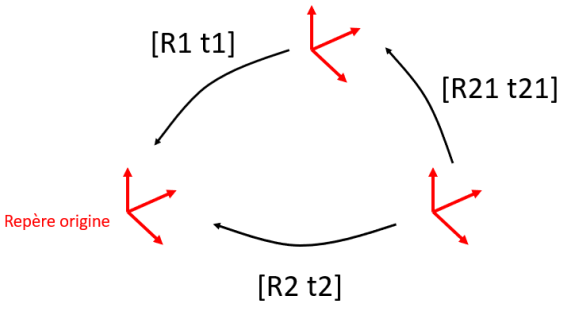


Figure 5 : Changement de vecteur

$$R21 = R2.R1^T$$

$$t31 = t2 - R2.R1^T.t1$$

Pour la mise en correspondance des points, il est possible d'utiliser la droite épipolaire qui permet de déterminer le point correspondant dans la deuxième image. Or dans ce cas nous avons à notre disposition des images déjà rectifiées. Il n'est donc pas nécessaire de calculer la droite épipolaire. Il suffit de faire une corrélation 2D selon la même ligne. Dans le cas ci-dessous, on souhaite faire la correspondance d'un point entre 2 images. Ce point appartient à la même ligne. Ainsi on peut tracer une droite qui coupe horizontalement l'image. Ensuite, on va parcourir cette droite en faisant une corrélation 2D avec l'image de référence. On pourra choisir la taille du masque qui correspond à la matrice blanche sur le dessin. En parcourant la droite avec la matrice blanche, on va garder en mémoire le maximum du résultat de la corrélation. Lorsque ce nombre est maximal soit 1, cela veut dire que les deux images sont identiques. Ainsi, en jouant sur le seuil de corrélation et le masque, on trouve le point correspondant à l'endroit ou la corrélation est maximale. On pourra donc savoir sa position exacte. Le pixel sera sur la même ligne. La colonne sera située au maximum de la corrélation.

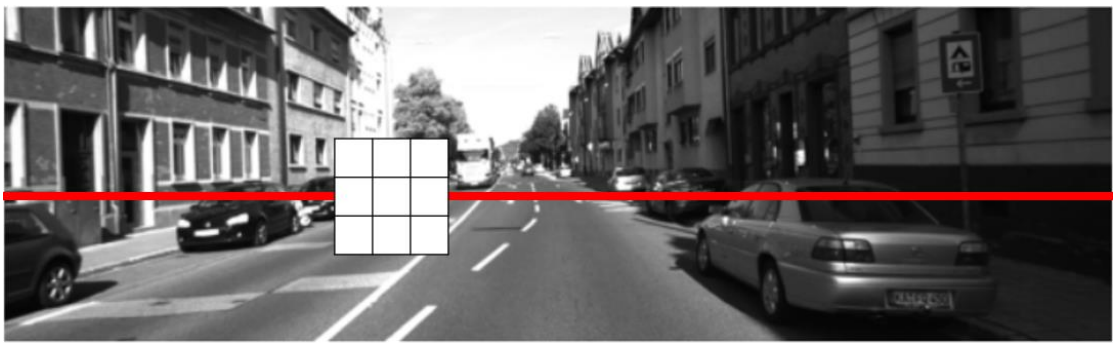


Figure 6 : Mise en correspondance

Maintenant que l'on a mis en correspondant les points sur les deux images, il nous reste à calculer la réelle position de la voiture. Dans notre cas nous avons les positions en pixels. On cherche la position du point $P = (X, Y, Z)$.

B. Triangulation

La triangulation et une méthode pour calculer une position à partir de deux points dont on connait la distance.

Les point A et B sont des positions que l'on connaît et le point C est la position que nous cherchons à déterminer.

Comme nous connaissons les positions des points A et B nous pouvons calculer la distance AB. Et en mesurant les angle α et β nous pouvons calculer ɣ le troisième angle du triangle ABC

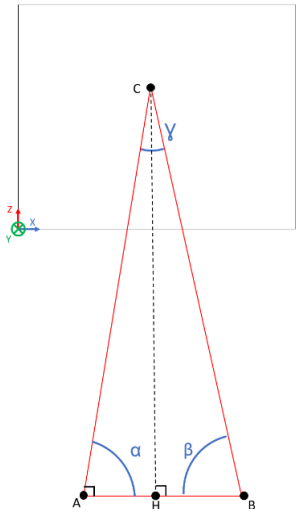


Figure 7 : La triangulation

Avec :

$$\gamma = 2\pi - \alpha - \beta$$

Ainsi avec la loi des sinus :

$$\frac{AB}{sin(\gamma)} = \frac{BC}{sin(\alpha)} = \frac{CA}{sin(\beta)}$$

$$BC = \frac{AB}{sin(\gamma)} \times sin(\alpha)$$

Et nous pouvons calculer la distance HC et BH avec sinus et cosinus de BC :

$$HC = BC \times sin(\beta)$$
$$BH = BC \times cos(\beta)$$

Nous pourrons donc calculer la position du point C.

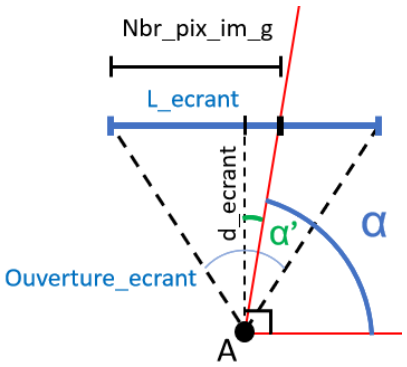


Figure 8 : Calcule de l'angle α

Pour calculer les angle α nous allons nous passer dans ce schéma :

Les seules données que nous avons sont : L'angle d'ouverture (ouverture_ecran) de la caméra le nombre de pixel en longueur de l'écran (L_ecran) et le nombre de pixels séparant le bord de l'image et le point C sur cette image (Nbr_pix_im_g).

Calculons déjà la d_ecran (distance en pixel) :

$$d\_ecran = \frac{L\_ecran/2}{tan\left(ouverture\_ecran/2\right)}$$

Puis nous pouvons calculer l'angle α' avec :

$$\alpha' = arctan\left(\frac{abs(nbr\_pix\_im\_g - L\_ecrant/2)}{d\_ecran}\right)$$

Donc nous nous pouvons calculer l'ange α avec :

$$\alpha = \frac{\pi}{2} \pm \alpha'$$

On fait de même avec l'angle β et nous pouvons calculer les coordonnées de C comme vu précédemment (*figure 9*).

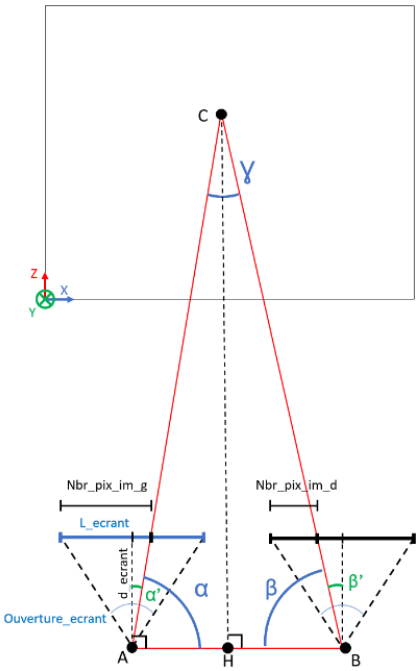


Figure 9 : Calcule de l'angle β

C. Affichage dans un plan 2D

Pour afficher les positions des voitures calculer par rapport au véhicule il faut ajuster les distances calculées. L'écran est une distance fictive il faut donc la soustraire et il faut soustraire la position de la caméra 3 (point B) par rapport au point d'origine.

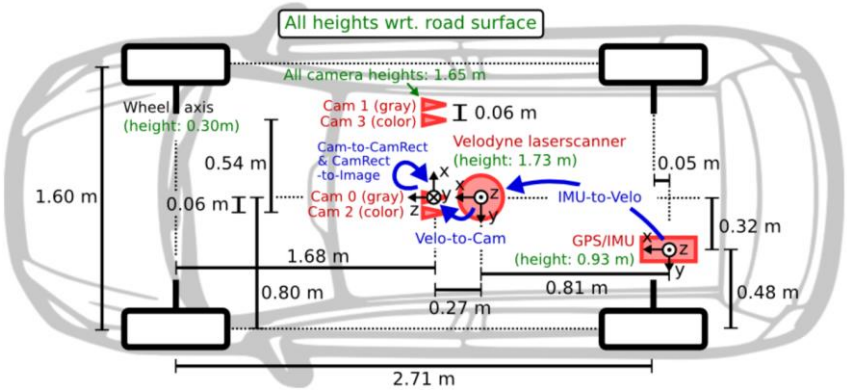


Figure 10 : Schéma de la position des caméras

```
d_X=-np.cos(beta)*BC+T3[0]
d_Z=np.sin(beta)*BC-T3[2]-d_ecrant*4.65*10**(-6)
```

Figure 11 : Formule des distances selon X et Z

D. Test de la mise en correspondance et de la triangulation

Avant de l'associer avec le travail réaliser dans la partie 1 et 2, on ajoute directement la position en pixel des voitures sur l'image de référence pour ensuite faire la mise en correspondance sur la 2ème image. On travail donc pour l'instant sur une seule image.

Ci-dessous, voici les résultats de la mise en correspondance (*Figures 12* et *13*) puis de la position des voitures (*Figure 14*).

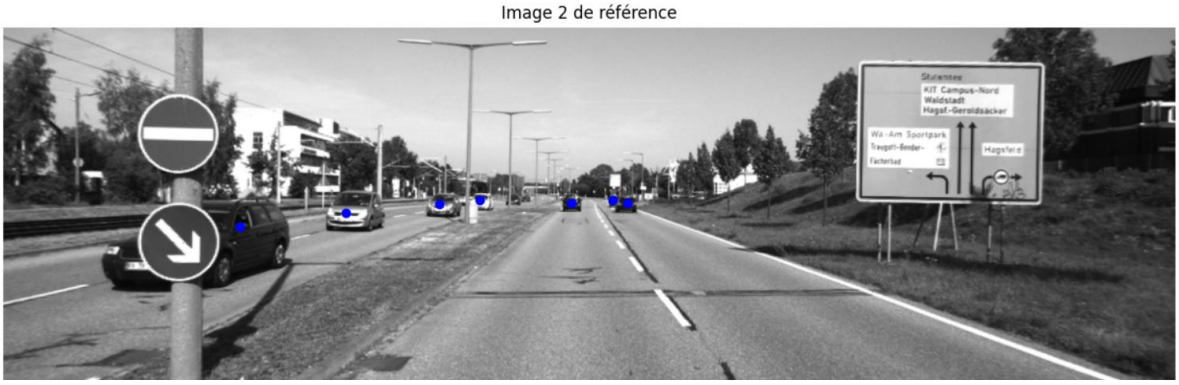


Figure 12 : Image de référence avec la position des voitures entrée à la main

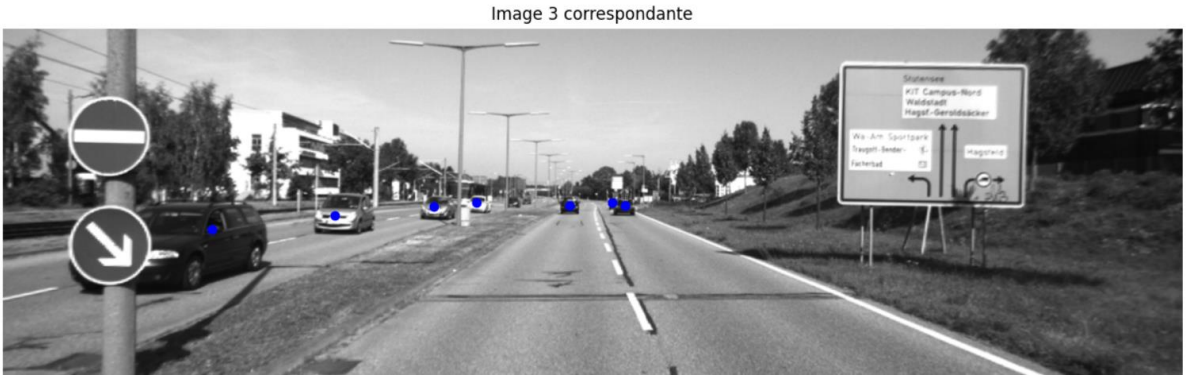


Figure 13 : Image correspondante avec la position des voitures trouvées

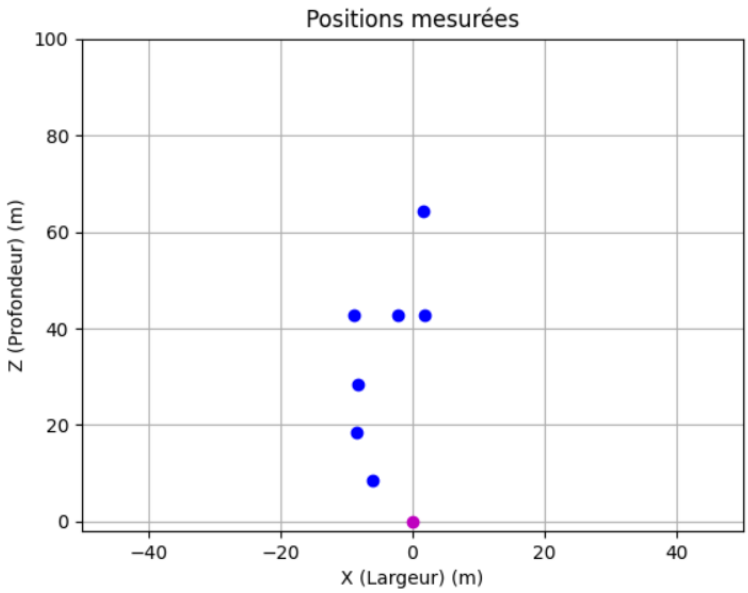


Figure 14 : Position des véhicules

Conclusion

L'objectif de ce projet était de créer une application en langage Python permettant la détection, le suivi et la localisation 3D de véhicules vus par une tête stéréoscopique dans une situation de circulation routière.

Pour cela, on s'est appuyé sur la bibliothèque OpenCV et la base de données fournie par le projet KITTI. On a notamment utilisé l'algorithme de Viola & Jones pour la détection des voitures, l'algorithme MedianFlow pour le suivi de ceux-ci et de la triangulation pour leur localisation en 3D.

Le résultat final présente un comportement correct avec une détection et un suivi efficace des voitures environnantes, tout en représentant leur position reconstruire en vue du ciel.

Le projet est disponible sur GitHub à l'adresse : https://github.com/LucasISSARD/vision-artificielle