ENSTA Bretagne
2, rue François Verny
29806 BREST cedex
FRANCE
Tel +33 (0)2 98 34 88 00
www.ensta-bretagne.fr

Middleware

UE 4.1

5 avril 2020

ROS : implémentation du suivi de ligne sur une voiture à l'échelle 1/10ème

Colin Baumgard, Ludovic Diguet, Hamid Hacene, Corentin Lemoine, Antonin Lizé

Table des matières

1 Introduction

Ce projet s'inscrit dans l'UE 4.1 : Middleware. Il s'agit d'implémenter sur une voiture à l'échelle 1/10ème des algorithme d'autonomie afin d'accomplir une mission : faire le tour du terrain de rugby de l'ENSTA Bretagne.

Ce rapport d'avancement présente brièvement le travail effectuer par l'équipe "LaCJCROS". La répartition du travail au sein du groupe peut être retrouver sur le dépôt git à l'adresse suivante : LaCJC

1.1 Le cahier des charges

Fonction	Désignation	Critères	Niveau	Fléxibilité
FP1	Permettre que la voiture fasse un tour de circuit en autonomie	Vitesse Précision max par rapport au milieu de la ligne	10km/h 5cm	F3: ±5km/h F1: +1cm
FC1	Doit avoir un poids adapté	Poids	4kg	F3
FC2	Doit permettre la maîtrise de la vitesse	Commande PWM		F0
FC3	Ne doit pas consommer trop de batterie			F2
FC4	Doit gommer les aspérités du terrain	Souplesse de la structure et des composants		F2

FIGURE 1 – Cahier des charges

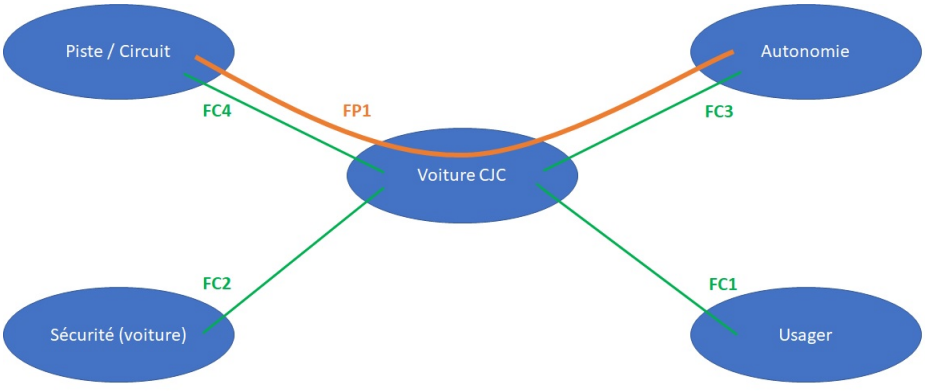


FIGURE 2 – Diagramme pieuvre

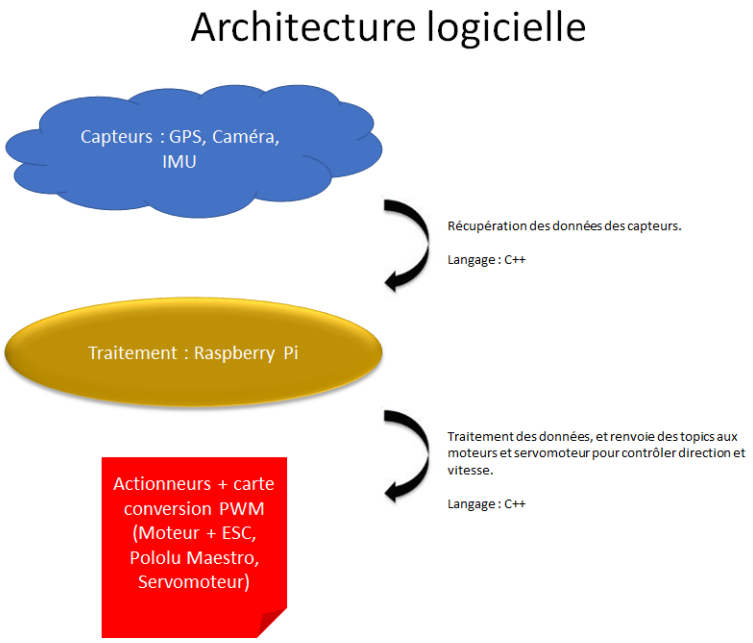


FIGURE 3 – Architecture Logicielle Matérielle

L'architecture C2 sera présentée une fois les différentes briques du projet testées et admises. Ce choix est fait par l'équipe de développer les packages séparément pour avoir moins de conflits à gérer et surtout pour assurer un maintien opérationnelle du code de chaque partie.

1.2 Les pistes envisagées

Différentes pistes pour rendre la voiture autonome ont été travailler : GPS, centrale inertielle et caméra. Dans un premier temps, le GPS et la centrale IMU ne seront pas utiliser pour accomplir la mission (faire un tour du terrain).

En effet, un bon traitement d'image suffit pour résoudre le problème. Il n'est pas nécessaire de se localiser dans l'espace pour suivre une route bien délimitée. Cette stratégie peut s'avérer utile dans des cas extrêmes, par exemple : perte de la ligne de la route, trop de vibrations liées à une route accidentée... Ce point fera l'objet d'une étude approfondie ultérieurement. Le concept de "Data fusion" sera exploré.

2 Structure du projet

2.1 Simulation

Vu les événements récents, nous utiliserons une simulation de la voiture sur *V-REP* construite en cours de *simulation de robots mobiles* avec M.*Zerr* (tutoVrep).

Pour cela, la voiture a été modélisée et une scène modélisant le terrain de rugby a été reconstituée
Une caméra a été modélisée également sur la voiture. Les codes de communication avec *ROS* ont été développé en *lua*. La communication se fait à travers les trois topics suivants :

- /image : pour la publication de l'image de la caméra ;
- /vrep_steer_angle : std_msgs/Float32, pour gérer la direction ;
- /vrep_speed_motor : std_msgs/Float32, pour régler la vitesse de rotation du moteur.

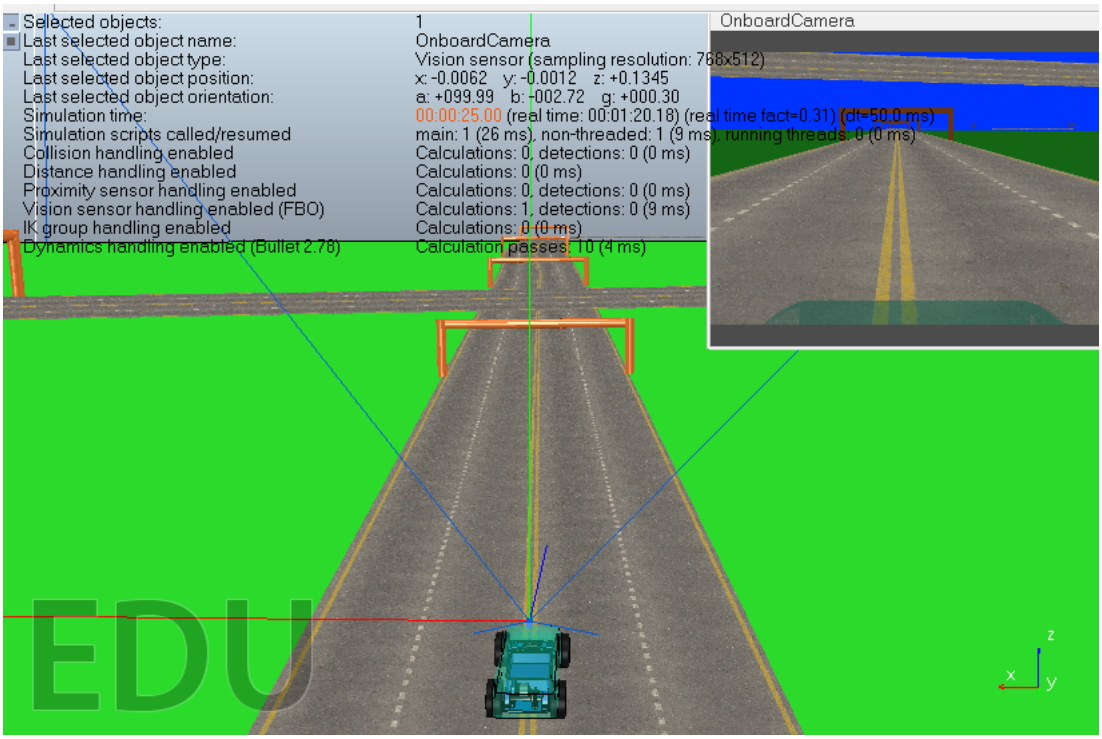


FIGURE 4 – Simulation sous *V-REP*

2.2 Drivers

Les drivers sont implémentés en utilisant le plus possible les nodes distribuées avec ROS. Le driver du GPS est celui du package *nmea_navsat_driver* ; la node utilisée est *nmea_serial_driver*. La node *navsat_transform_node* du package *robot_localization* transforme alors la donnée issue du driver en une projection dans le repère du robot.
Le package *cv_camera* nous permet de publier une image sur un topic ROS issue de la caméra de la Raspberry pi.
Pour l'envoi de signaux PWM vers l'ESC et le servo de direction, on a codé une node Python utilisant une bibliothèque pour contrôler la carte Maestro via la connexion série USB.

2.3 Traitement d'image

Pour la partie traitement du flux de la caméra, deux solutions se sont présentées :

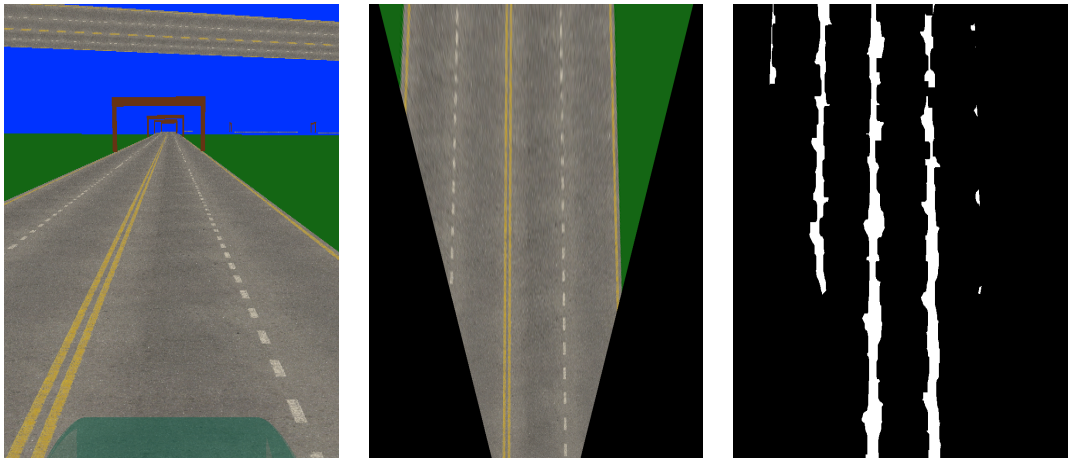
- **Suivre la ligne du milieu de la route** : cette approche est assez simple dans le sens ou avec des opérations morphologiques simple et avec une méthode de détection de contours basique (comme celles étudiées en cours de TNI), on peut calculer l'erreur relative du centre de notre image avec la ligne à suivre, et ainsi corriger la trajectoire avec un contrôleur *PID* classique. Le problème de cette méthode est qu'elle n'est pas réaliste : les algorithmes ainsi développés ne peuvent être utiliser pour appréhender une autonomie en milieu urbain (sur une vraie route). C'est pour cela que nous avons choisis une autre approche ;
- **Contrôler le véhicule par rapport à la courbure des deux lignes constituant la voie qu'emprunte la voiture**. Cette méthode peut s'avérer très robuste car elle permet de prendre en compte le cas ou la voiture change de voie par exemple.

Pour implémenter cette méthode, nous devons suivre les étapes suivantes :

1. Calculer la distorsion de la caméra ;
2. Appliquer une correction de distorsion aux images en sortie ;
3. Calculer la *BirdEyeView* de l'image : c'est une projection qui permet de cibler une zone d'intérêt (devant la voiture) et de calculer une projection vers un point artificiel (vue du dessus). Ceci permet de supprimer l'effet d'horizon et de garder les lignes parallèles comme telles ;
4. Utiliser des transformations morphologiques pour filtrer l'image et extraire les zones d'intérêt ;
5. Détecter les pixels de la voie et ajuster pour trouver la limite de la voie ;
6. Déterminer la courbure de la voie en utilisant la méthode *Sliding Window Search* et une interpolation polynomiale ;

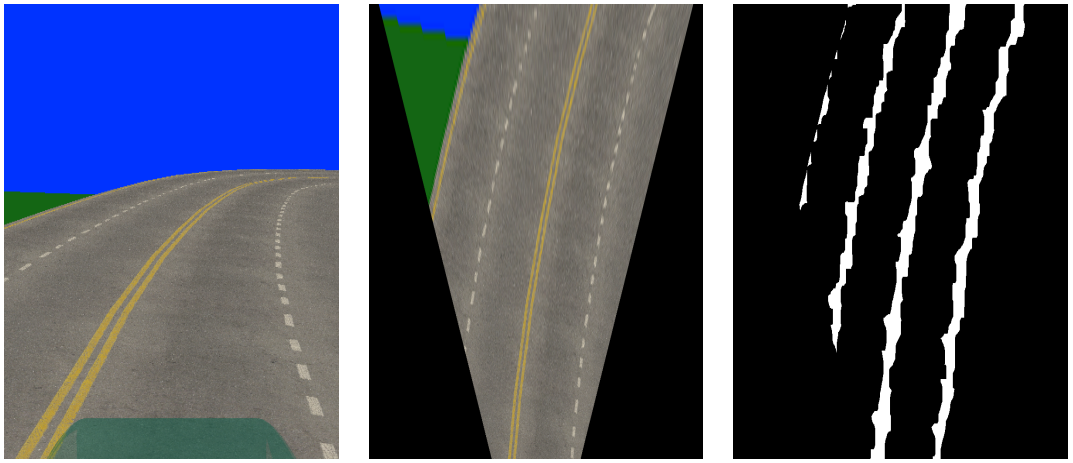
7. Calculer la position du véhicule par rapport au centre et trouver l'erreur à renvoyr pour le *PID*.

Nous sommes qu'au début du développement de ces algorithmes, mais les résultats sont prometteurs. A souligner que cette méthode est beaucoup utilisée pour la conduite autonome de voitures.



(a) Source image (b) Bird Eye View (c) Lane detection

FIGURE 5 – Traitement sur un segment droit



(a) Source image (b) Bird Eye View (c) Lane detection

FIGURE 6 – Traitement sur un virage

Contrôleur

Lorem ipsum dolor sit amet

Exemple latex langage c :

```
__kernel void mandelbrot(__global float2 *q,
__global ushort *output, ushort const maxiter)
{
    int gid = get_global_id(0);
    float nreal, real = 0;
    float imag = 0;
    output[gid] = 0;
    for(int curiter = 0; curiter < maxiter; curiter++)
    {
        nreal = real*real - imag*imag + q[gid].x;
        imag = 2* real*imag + q[gid].y;
        real = nreal;
        if (real*real + imag*imag > 4.0f)
            output[gid] = curiter;
    }
}
```

3 Installation ROS sur Raspberry

Lorem ipsum dolor sit amet coming Soon!!

4 CAO

Lorem ipsum dolor sit amet coming Soon!!

5 Docker pour réutilisation

Lorem ipsum dolor sit amet coming Soon!!

6 Conclusion

Lorem ipsum dolor sit amet coming Soon!!