



Rapport Projet Thématique PR214 :

NXP Cup



MOUHAGIR Ayoub
OULED-AMEUR Amjad

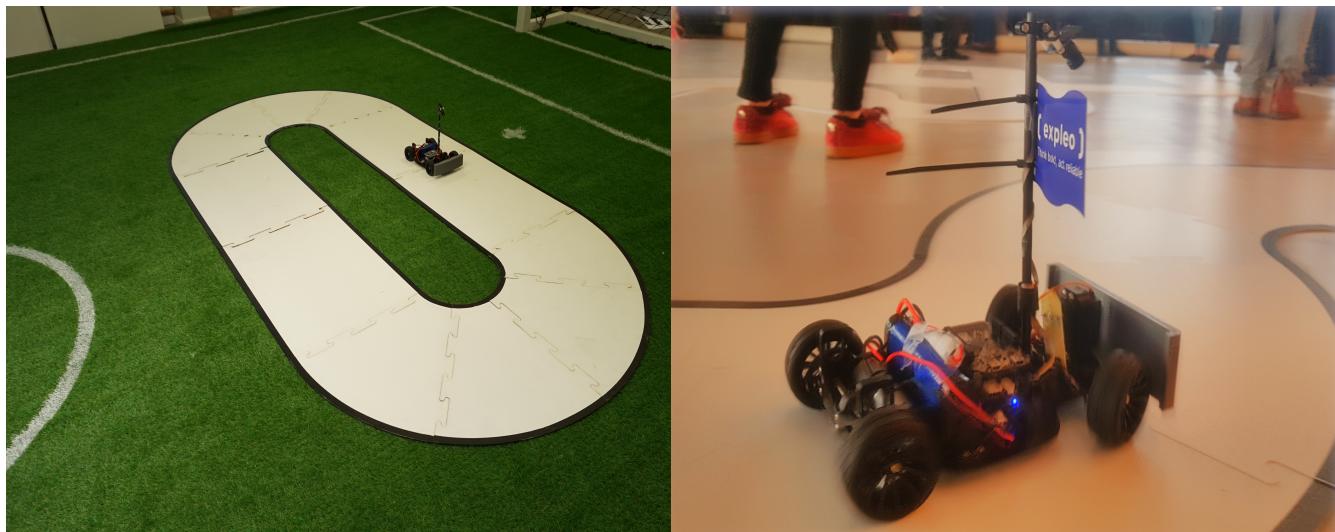
Encadré par :
Guillaume Ferré

Table des matières

1	Introduction	3
2	Kit de démarrage NxP	3
3	Spécifications matérielles	4
4	Programmation du micro-contrôleur	7
5	Programmation en Simulink	8
5.1	Caméra	8
5.2	Servomoteur	9
5.3	Moteurs à propulsion	10
5.4	Boutons pousoirs	10
6	Algorithmes de traitement d'image :	11
7	Direction du servomoteur	14
8	Commande des moteurs à propulsion	15
9	Boutons Poussoirs	17
10	Quelques autres améliorations sur les performances du véhicule	18
10.1	Point mort de la direction (Dead Spot)	18
10.2	Contrôle de la direction par les roues arrières	18
10.3	Calibrage de la luminosité de la caméra	19
11	Modèle complet	20
12	Communication bluetooth	21
13	Conclusion	22
14	Références :	23

Remerciements

Nous tenons à remercier notre école et les membres de l'équipe NxP Cup qui nous ont permis de participer à cette aventure formidable que représente ce challenge. En particulier notre encadrant Mr. Guillaume Ferré qui s'est toujours démené pour trouver des solutions que ce soit pour nous aider à résoudre des problèmes techniques, logistiques ou organisationnels.



1 Introduction

La NxP Cup est une compétition mondiale organisée chaque année qui réunit plusieurs équipes d'étudiants et clubs de robotique pour construire, programmer et piloter, sur circuit, un modèle réduit de voiture autonome. Et ce n'est pas qu'une affaire de « fun » : l'équipe gagnante en qualifications mondiales s'envolera pour la Silicon Valley!. La finalité de la NxP Cup est de construire et piloter des mini-voitures autonomes qui devront parcourir un circuit le plus rapidement possible. Le trajet n'est pas connu à l'avance (Il peut y avoir des croisements, des ponts, des ralentisseurs, des tunnels ...) et la voiture proposée par les organisateurs est équipée des moteurs, des cartes de puissances, d'une carte micro-contrôleur et d'une caméra pour détecter les bords de la piste.

2 Kit de démarrage NxP

Jusqu'en 2017, le seul modèle autorisé à la compétition était le "Model C" présenté dans la figure 2. Et depuis cette année un nouveau modèle était adopté par la Nxp qui est le modèle "Alamak" (Figure 3). Ces deux modèles ont les mêmes fonctionnalités de base avec des légères différences au niveau de manipulation des cartes. Ces châssis comportent 2 moteurs à balais 7.2v maximum pour la propulsion, une batterie et un servomoteur standard pour la direction. En plus des cartes d'interface de puissance pour piloter le servo et les moteurs.



FIGURE 2 – Modèle C



FIGURE 3 – Modèle Alamak

Au cours de notre préparation pour la compétition, on a adopté un modèle de voiture hybride des deux modèles proposés (Châssis et moteurs de l'Alamak et les cartes du "Model C") comme les cartes du "Model C" comportent plusieurs I/O que l'autre modèle (Potentiomètres, boutons et des LEDs). Cela nous a permis de pouvoir changer les paramètres de la voiture manuellement, notamment la vitesse des moteurs et la position des roues de devant. Malheureusement, à la dernière ligne droite pour la compétition, on avait un problème dans l'une des cartes de pilotage. Alors, on était obligé de reprendre le modèle d'Alamak.

3 Spécifications matérielles

Dans cette partie, nous décrivons les différents composants du Kit de démarrage :

- **1xLine Scan Camera** : c'est un module de caméra à balayage linéaire, il se compose d'un réseau de capteurs linéaires CMOS de 128 pixels et d'un objectif ajustable. Cette caméra a une résolution de 1x128. Elle permet de collecter les luminosités d'un nombres de points d'un segment délimité par l'angle de la caméra et la configuration de son objectif, ces luminosités sont ensuite convertis en tensions analogiques.
- **1x Servo Motor** : c'est le module permettant de piloter et d'asservir en angle d'orientation les deux roues d'avant de la voiture. En effet, l'angle de rotation est proportionnel à la tension de commande.
- **2x moteurs à propulsion** : c'est 2 moteurs pilotent les roues d'arrière de la voiture. Ceci est un avantage permettant d'appliquer des commandes différentes à chacune des roues ce qui donne la possibilité de passer en mode différentiel. Le signe de la tension de commande du moteur détermine son sens de rotation. De plus, la tension de commande est analogique ce qui permet de gérer la vitesse du moteur associé.
- **Carte du micro-contrôleur μC** : cette carte embarque le μC KL25Z, le module OpenSDA permettant l'implantation et le debug du code à implémenter, un accéléromètre xyz qui peut être utilisé pour détecter si la voiture est sur une pente ou si la voiture n'avance plus (cas de collision), et plusieurs I/O permettant le pilotage des capteurs/actionneurs de la voiture.

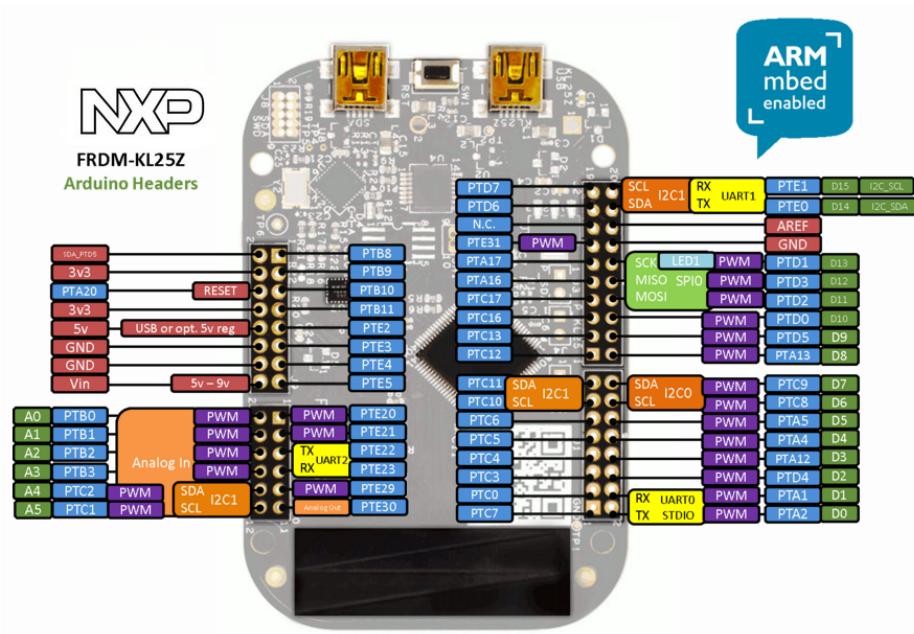


FIGURE 4 – La carte du micro-contrôleur

- **Carte de capteurs/actionneurs modèle Alamak :** c'est la carte de correspondance entre celle du μ C et les différents capteurs/actionneurs, on y trouve : deux modules permettant de gérer deux servos moteurs différents, deux modules permettant de gérer deux caméras de type lineScan, un module bluetooth et 5 boutons poussoirs permettant ultérieurement des réglages manuels. ces composants sont bien illustrés ci-dessous :

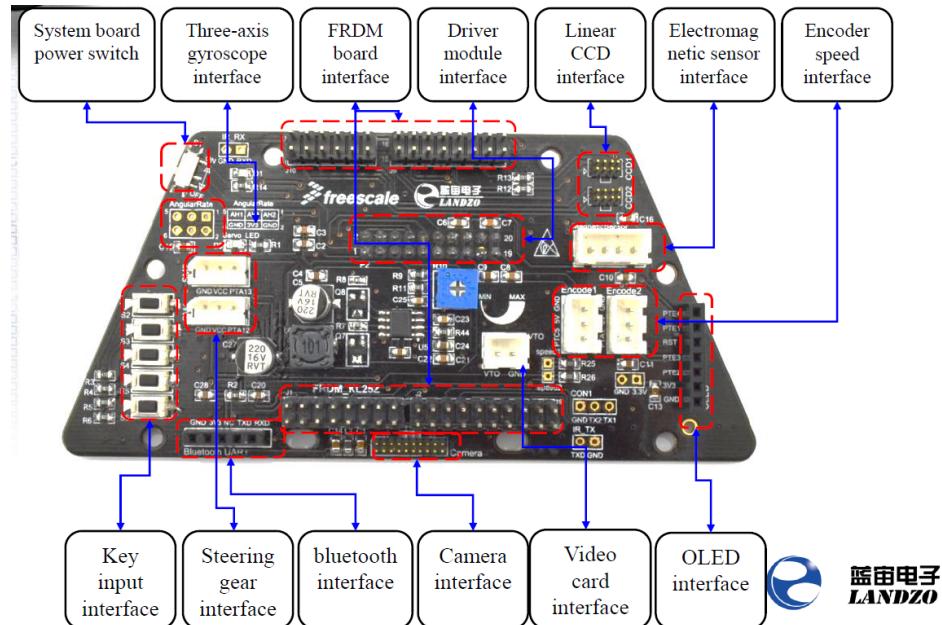


FIGURE 5 – La carte de capteurs-actionneurs

- **Carte de puissance modèle Alamak :** cette carte permet d'alimenter tous les composants de la voiture, sa source d'alimentation est principalement une batterie dont

l'énergie est adaptée et distribuée par le biais de cette carte de puissance. Cette carte alimente également les moteurs à propulsion de la voiture par une tension analogique et un courant assez fort vis-à-vis ces moteurs.

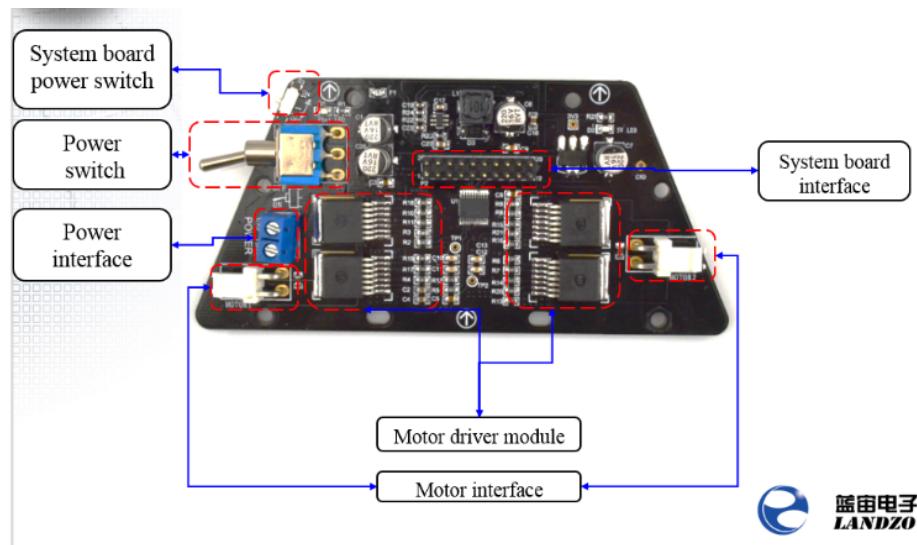


FIGURE 6 – La carte de puissance



4 Programmation du micro-contrôleur

Quant à l'aspect programmation du μC , plusieurs solutions sont possibles et chacune présente des avantages et des inconvénients :

- **programmation en C** : ce langage a l'avantage qu'il est proche du langage machine, ceci permet une meilleure gestion de ressources matérielles telles que les mémoires (allocations dynamiques ...) et la vitesse de traitement (gestion directe des interruptions et des fonctionnalités d'accélération de calculs offertes par le μC comme la vectorisation). La contrepartie de la programmation en C est qu'il est moins intuitif et demande une base solide en programmation embarquée. Pour programmer en C, vous pouvez installer codeWarrior Special Edition dont une licence n'est pas nécessaire.
- **Programmation via Matlab** : le module Simulink de Matlab permet de commander et piloter intuitivement les différents actionneurs de la voiture, il permet aussi de faire des acquisitions en temps en réel de n'importe quelle structure de donnée. La contrepartie de Simulink est qu'il gère implicitement les ressources matérielles du μC , ceci nous élimine des degrés de liberté sur l'optimisation du programme à implémenter. Afin de programmer en Simulink, vous pouvez demander une licence Matlab dans le cadre de la compétition, puis installer l'add-on NXP KL25Z et le package NXP CUP COMPANION.
- **Programmation via Mbed** : L'avantage de Mbed c'est qu'il inclut des librairies prédéfinies facilitant la programmation. Parmi ces librairies, nous trouvons celle qui gère le module Bluetooth (établissement de la connexion, émission/réception de données).

Conseil sur choix de l'interface de programmation : Nous proposons aux prochains candidats de commencer par Simulink pour tester les différents composants de la voiture (caméra, servomoteur, moteur à propulsion). En effet, il faut tout d'abord être capable d'envoyer les données de la caméra au travers du port série à l'ordinateur pour le visualiser en temps réels, puis après on peut essayer d'afficher d'autres informations qui peuvent être utiles (ex : sortie d'un filtre appliqué aux données issues de la caméra). Nous proposons ensuite de tester des algorithmes de traitement d'image afin de les comparer et de noter les avantages et les inconvénients de chacun. Lorsque vous aurez fait vos tests de traitement d'image, commande des servomoteurs, acquisition de l'état des différents boutons pousoirs et la commande des moteurs à propulsions, vous pouvez envisager de passer en C pour bénéficier de ses performances indéniables.

5 Programmation en Simulink

Dans cette partie, on détaillera notre approche de programmation en partant au début sur les cartes du modèle C, puis de l'adapter au modèle Alamak. Cette programmation sur Simulink était notre principale approche dans la compétition vu le temps restreint donné pour accomplir la tâche. Malgré, des essais de programmer en C pour gagner au terme de performances qui n'étaient pas allés jusqu'au bout.

Après avoir monté tous les composants de la voiture et vérifié qu'ils sont tous bien alimentés, on essaiera de comprendre les éléments de base pour faire une voiture autonome. Il existe trois éléments : Direction, propulsion et véhicule enabling. Dans un premier temps, on regardera le premier composant important pour le système de direction qui est le fait d'obtenir les données de la caméra et de les traiter.

5.1 Caméra

La caméra renvoie un vecteur ligne de 1x128 valeurs qui représentent la luminosité le long d'une ligne de mesure. Afin de visualiser les données de la caméra, nous proposons le circuit ci-dessous :

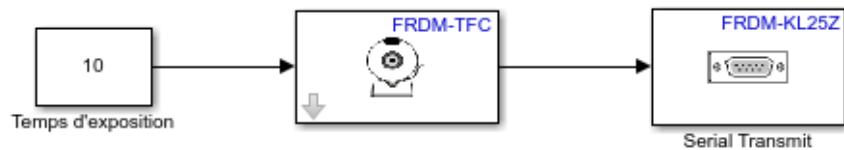


FIGURE 7 – schéma fonctionnel de l'asservissement

En effet, le temps d'exposition est un élément crucial vis-à-vis la précision et la stabilité de l'acquisition. En effet, l'exposition est la quantité de lumière reçue par chacun des capteurs de la caméra. Si le capteur reçoit trop de lumière, l'image sera sur-exposée c'est à dire avec des zones très blanches dénuées de détail. Si le capteur ne reçoit pas assez de lumière, la photo est dite sous-exposée : des zones sont noires, également dénuées de détails. Le but est donc de trouver le juste milieu pour perdre le minimum de détail.

De plus, le bloc de la caméra vous donne la possibilité de choisir l'une des deux caméras et de fixer le temps d'échantillonnage des données de la caméra. Bien entendu, le temps d'échantillonnage doit être supérieur que le temps d'exposition, sinon on se place dans le cas où on demande à la caméra de nous délivrer des données qui ne sont pas encore prêtées. Enfin, la sortie de la caméra est visualisé par le bloc Serial Transmit, dans ce cas vous devriez choisir une transmission en "printf" et non pas en "putc".

Pour visualiser les données, vous pouvez utiliser le module nxp cup companion sur Matlab. si l'on met la voiture sur un fond blanc délimité par deux bandes noires, on obtient le résultat suivant :

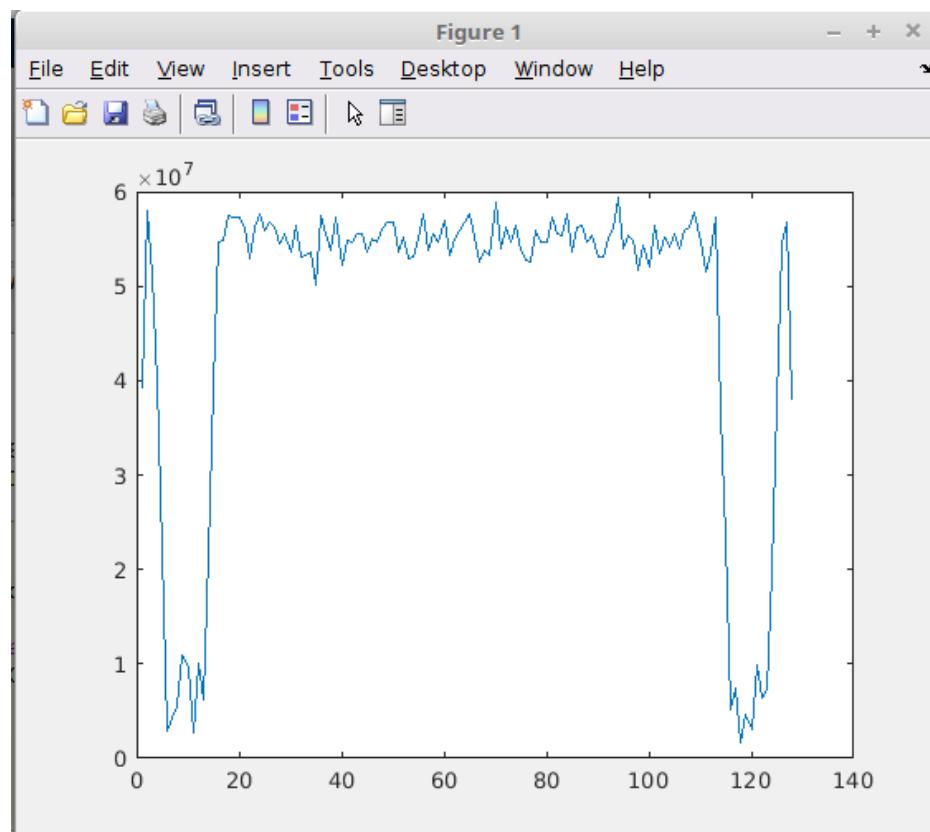


FIGURE 8 – données issues de la caméra

L'image que vous obtenez peut être bruité si le temps d'exposition est faible ou la luminosité environnante est insuffisante, ceci peut aussi causer des contours mouves ce qui peut gêner le traitement de cette image par la suite.

5.2 Servomoteur

La commande du servo moteur est simple mais doit respecter quelques règles pour ne pas endommager les roues d'avant :

- il vaut mieux d'essayer plusieurs commandes pour savoir les deux commandes de saturations correspondants à l'angle maximale et minimale de rotation des roues d'avant. Ceci vous servira aussi de déterminer l'offset de la commande, car une commande nulle ne correspond pas nécessairement à un angle de rotation nulle.
- la durée entre deux commandes successives ne doit pas être inférieur à 20ms (caractéristique du servo moteur fourni)

De plus, Le bloc permettant de commander le servo moteur est le suivant :

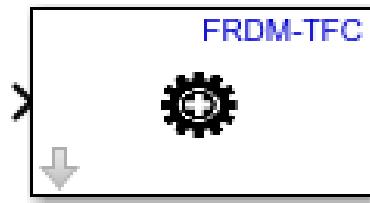


FIGURE 9 – bloc servo moteur

5.3 Moteurs à propulsion

La commande des 2 moteurs à propulsions est simple, chacun des 2 moteurs a un bloc de commande différent qui reçoit une valeurs entre -1.0 et 1.0 correspondant respectivement à une vitesse maximale dans un sens et la vitesse maximale dans le sens inverse, une commande nulle correspond à une vitesse nulle.

Le bloc permettant de choisir et de commande l'un des deux moteurs est le suivant :

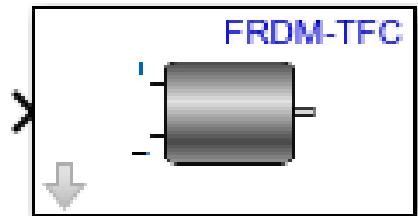


FIGURE 10 – bloc moteur propulsion

5.4 Boutons poussoirs

Les 5 boutons poussoirs dans la carte des capteurs/actionneurs sont connectés à une entrée analogique du micro-contrôleur par le biais d'un circuit analogique décrit ci-dessous :

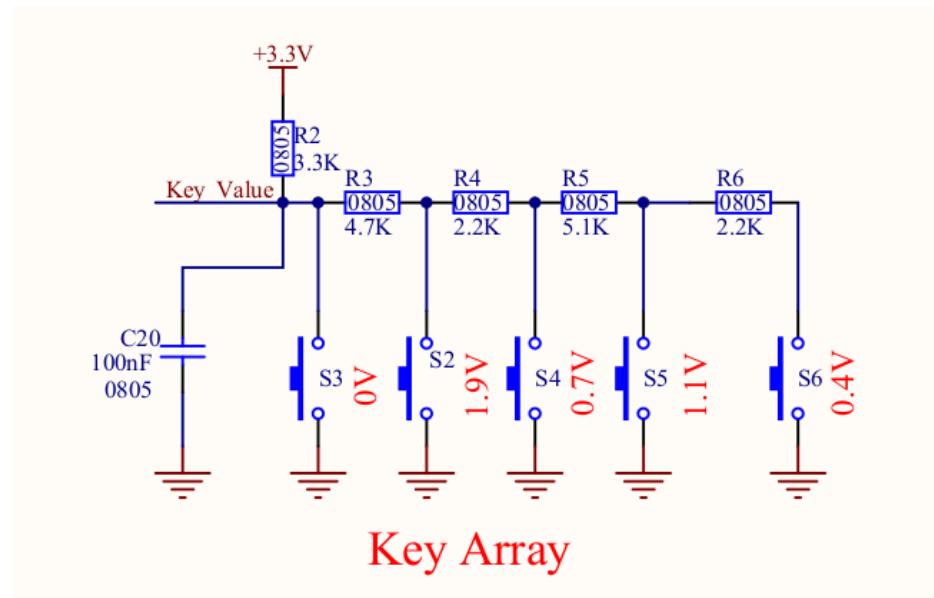


FIGURE 11 – schéma de connexion des boutons poussoir

Key Value est connecté à l'entrée PTC0 du micro-contrôleur (vous trouveriez les schémas de connections dans le répertoire github mentionné dans la section référence)

Dans notre cas, les valeurs des tensions analogiques mentionnées sur la figure ci-dessous ne correspondaient pas aux valeurs pratiques lorsque nous les avons affichées sur Matlab par le biais du port série. Par conséquent, il vaut mieux de faire des tests et relevé, sur Matlab, la valeur de PTC0 pour chaque bouton appuyé. Ceci vous permettrez d'identifier l'état des boutons poussoirs en comparant la valeur de PTC0 avec les valeurs que vous aurez mesurées initialement.

Le bloc permettant de lire la valeur d'une entrée analogique est le suivant :



FIGURE 12 – bloc entrée analogique

6 Algorithmes de traitement d'image :

Afin de détecter les deux bandes noires, il faut envisager un algorithme de traitement d'image efficace et flexible. Il vaut mieux respecter quelques contraintes sur l'algorithme à implé-

menter :

- Il doit être rapide. Le temps de traitement doit être inférieur aux temps d'échantillonnage de la caméra.
- Il ne doit pas être dépendant de la luminosité extérieure. En effet, la luminosité du jour n'est pas la même pendant le soir, donc pour le cas d'un tunnel l'algorithme doit s'adapter avec sa luminosité environnante.

Il existe plusieurs algorithmes possibles, parmi ceux :

- On peut envisager de prendre en référence l'intensité maximale. Puis, pour trouver la bande noir droite, on peut balayer le tableau des luminosités en commençant par le pixel de centre jusqu'au pixel 128, on sort de ce balayage si l'on trouve une valeur de luminosité inférieur à 75% de la référence calculée précédemment. Similairement, nous appliquant cet algorithme dans l'autre sens pour la trouver la bande noir gauche en commençant toujours par le pixel du milieu. Le résultat de ce programme est illustré ci-dessous :

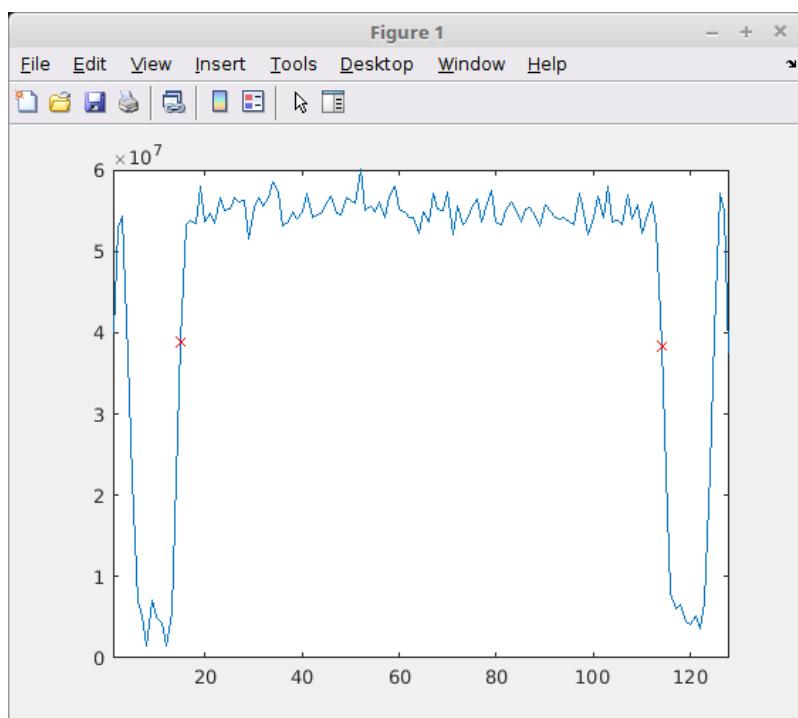


FIGURE 13 – résultat de l'algorithme du seuillage

Nous remarquons que ce programme détecte bien les deux bandes noires. Par la suite, on peut calculer le milieu de ces deux bandes noires et le comparer à la valeur 64, qui est bien le pixel de centre, pour savoir la position de la voitures par rapport à ces deux bandes noires.

Cet algorithme est simple et rapide, mais il n'est pas efficace pour des pistes non illuminées en homogène, on peut quand même jouer sur le seuil 75% pour avoir un résultat cohérent sur la plupart des cas.

- A part, le seuillage, on peut envisager une dérivée d'ordre 1 puis une détection des maxima locaux, on peut utiliser le filtre $[-1 \ 0 \ 1]$ approximant la dérivée d'ordre 1. Le résultat de cet algorithme donne le résultat suivant :

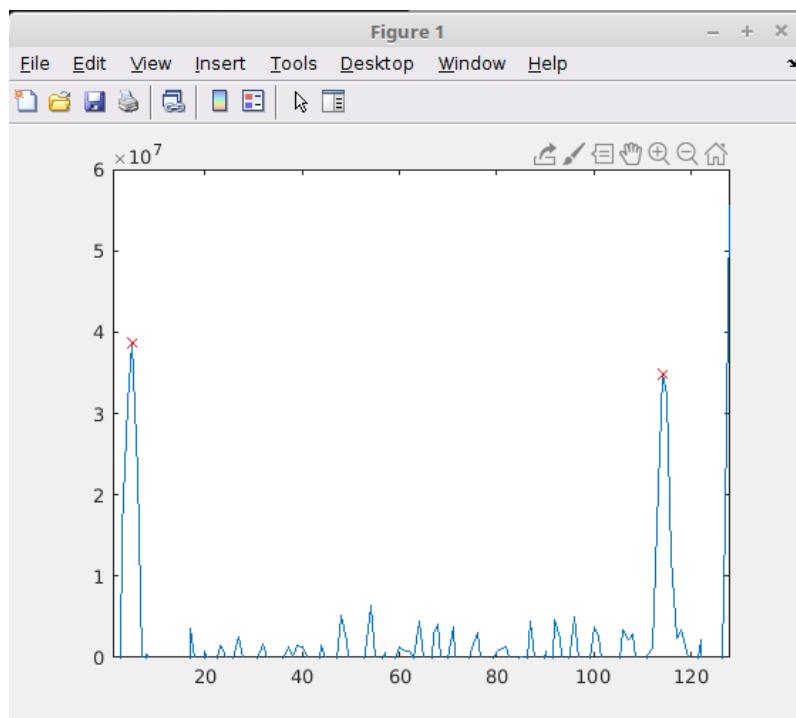


FIGURE 14 – résultat de l'algorithme de dérivation

Le résultat de cet algorithme est satisfaisant mais moins rapide que le premier. Toutefois, cet algorithme est plus flexible et polyvalent puisqu'il n'est pas basé sur un seuillage et donne des résultats moins aberrants dans le cas un milieu non homogène en terme de luminosité.

- Le dernier et non le moindre et le robuste filtrage de Canny. En effet, il s'agit de la dérivée de la gaussienne qui serait équivalent à une application simultanée d'un filtre gaussien et d'une dérivée d'ordre 1. Le résultat de ce filtrage est le suivant :

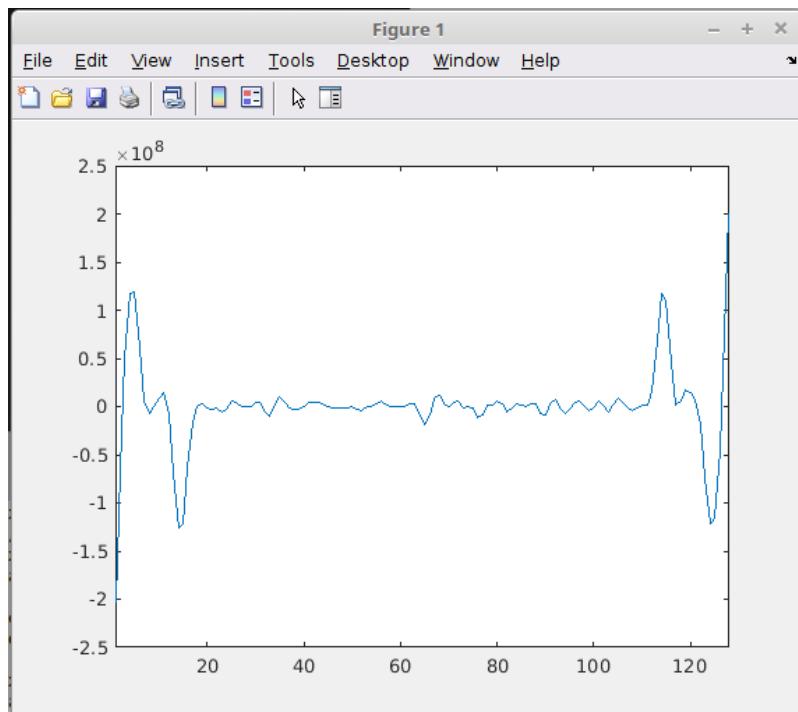


FIGURE 15 – résultat de l'algorithme du filtrage de Canny

Afin de détecter les indices des bandes noires, il faut détecter les extrema locaux (on peut appliquer le module sur le résultat du filtrage et raisonner uniquement sur les maxima locaux mais cela nécessite un temps de traitement de plus). Cet algorithme a l'avantage qu'il applique un filtrage gaussien sur l'image, ceci filtrera le bruit et lisserait la courbe. De plus, il permet de donner une information plus précise sur la position de la voiture par rapport aux bandes noires.

D'autre part, trois autres facteurs sont primordiales pour récupérer des images crédibles et nettes, se sont l'ajustage de l'objectif, angle et l'altitude de la caméra.

- Plus on augmente l'angle d'ouverture de l'objectif de la caméra, plus le segment visualisé par la caméra est large. Cependant, la taille des pixels va être plus grande ainsi on perd la précision.
- L'augmentation de l'angle et l'altitude de la caméra permettent d'anticiper les virages mais ça diminue la précision puisque la taille de chaque pixel serait plus grande.

7 Direction du servomoteur

Maintenant que nous avons un programme qui permet de détecter les bandes noires et renvoyer le centre de la piste. Nous pouvons créer un système qui ajuste la direction du servomoteur pour centrer automatiquement le véhicule. Dans cette partie, nous allons apprendre à tourner les roues du véhicule selon la donnée de sortie renvoyée par l'algorithme de LineScan. Cette donnée (indice entre 1 et 128) représente l'endroit où le véhicule pense le centre de la piste se trouve qui est relatif à l'endroit où le véhicule se trouve sur la piste. Ensuite, on définit le centre de la piste comme étant l'indice 64. Alors la différence entre l'indice 64 et l'indice du véhicule est notre signal d'erreur. On pourra calculer cet erreur à l'aide du modèle présenté dans la figure ??.

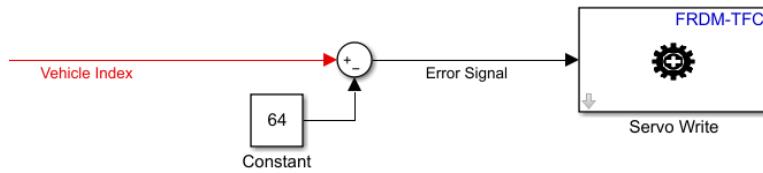
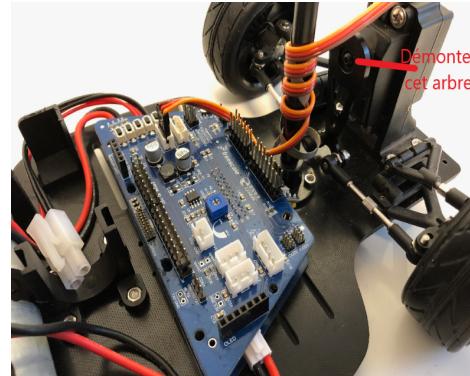
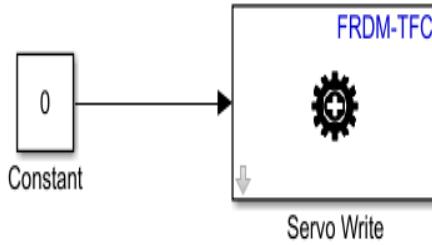


FIGURE 16 – Modèle du calcul de l’erreur

Notez que le signal d’erreur est envoyé directement au servomoteur pour contrôler la direction du véhicule. Nous avons défini le centre de la piste comme étant l’indice 64. Alors, si l’indice du véhicule est 64 (le véhicule se trouve au milieu de la piste) un signal zéro est envoyé au servomoteur ce qui force les roues d’être tout droit, ce qui n’est pas le cas au début. Pour régler ce problème, on démonte l’arbre de direction qui relie les roues avec le servomoteur et on envoie un signal zéro depuis Matlab au servo puis on remonte l’arbre de direction comme montré dans la figure ci-dessous. Ceci n’est pas la seule solution, on pourra aussi centrer nos roues manuellement à l’aide des boutons poussoirs (La manipulation de ces boutons sera détaillée dans la suite).



En outre, si l’indice du véhicule est supérieur/inférieur à 64, le signal d’erreur est positif/négatif ce qui fera tourner les roues dans un sens ou l’autre. Alors, il est nécessaire de vérifier que les roues tournent dans le bon sens en rapprochant le véhicule aux bandes noires. Si ce n’est pas le cas, il suffit multiplier l’erreur par -1 en rajoutant un bloc du gain négatif -1 après le bloc du calcul d’erreur. Jusqu’à présent, nous avons réussi à faire tourner les roues avant dans le bon sens pour déplacer le véhicule vers le centre de la piste. Vous avez probablement remarqué que les roues tournent, mais pas suffisamment. Nous pouvons résoudre ce problème en ajoutant un gain au système pour augmenter la taille du signal d’erreur. On pourra multiplier par exemple le signal d’erreur par 2. Ainsi, le signal transmis au servomoteur devient le double de ce qui était avant. Cela signifie que pour la même erreur, les roues avant tournent deux fois plus (Notez que si vos roues tournaient dans le mauvais sens dans la section précédente, ce gain devrait être de -2).

8 Commande des moteurs à propulsion

Maintenant que nous avons un système de direction qui marche parfaitement et que notre véhicule peut rouler sur la piste lorsqu'il est poussé. Il est temps de laisser la voiture se déplacer de sa propre énergie. On commence par l'envoi d'une constante au commandement des moteurs, ce qui nous permet de connaître l'ordre du paramètre de la vitesse du véhicule (entre 0 : No power et 1 :full

power). Alors, pour modifier la vitesse, on modifie la constante et on recompile le programme. Avec cette vitesse constante, nous pouvons effectuer un certain nombre de tests pour voir les effets de divers paramètres tels que le gain de direction du servomoteur et l'angle de la caméra.

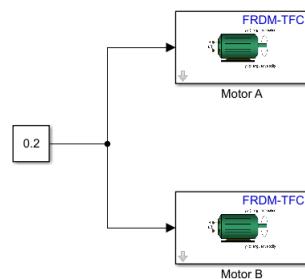


FIGURE 18 – Ajout du commande des moteurs à propulsion avec vitesse constante

Nous avons maintenant une idée de la façon dont la vitesse du véhicule et l'angle de la caméra affectent les performances du véhicule. Nous allons mettre la vitesse comme un paramètre qu'on augmente ou diminue manuellement avec les boutons sans besoin de compiler à chaque fois à l'aide des boutons poussoirs.

Une chose que vous avez peut-être remarquée, c'est que la vitesse du notre véhicule est limitée par les virages. Sur les lignes droites, on peut rouler très vite, mais pour faire un virage, il faut aller beaucoup plus lentement. A vitesse constante, il faut régler la vitesse à un niveau bas pour pouvoir tourner. Cependant, cette vitesse lente semble assez lente lorsque la voiture est sur une ligne droite. Une solution simple consiste à définir la vitesse en fonction de l'angle de braquage. Lorsque l'angle de braquage est faible (comme sur une ligne droite), nous pouvons avoir une vitesse élevée. Lorsque cet angle est important (sur un virage), nous pouvons réduire la vitesse. Pour ce faire, on se sert du signal envoyé au servomoteur qui représente l'angle de braquage en temps réel. Nous pouvons implémenter une fonction mathématique en Matlab qui prend la valeur absolue de cet angle comme argument et renvoie la vitesse :

- Si cette valeur est comprise entre 0 et 10 par exemple, la vitesse sera à 100%.
- Sinon, c'est elle est plus grande, la vitesse doit être réduite linéairement par exemple de 100% à 50%.

Sur Simulink, il existe des blocs qui permettent de concevoir ces fonctions mathématiques linéaires. Ceux sont des "lookup tables" présentées dans la figure 19. Elles se comportent comme des fonctions linéaires avec des pentes différentes d'un segment de valeurs à autre.

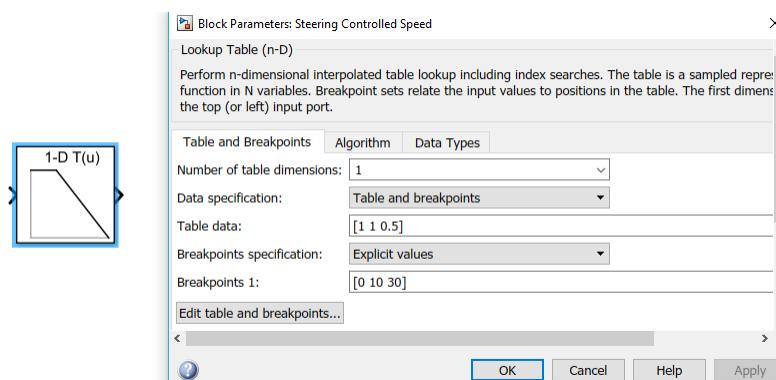


FIGURE 19 – Conception d'une Lookup table

Cette méthode nous permettra de réduire la vitesse du véhicule aux virages tout en conservant une vitesse maximale sur les lignes droites. Le modèle qui commande les moteurs (avec l'implémentation de modification de la vitesse avec les boutons poussoirs) est présenté dans la figure 20.

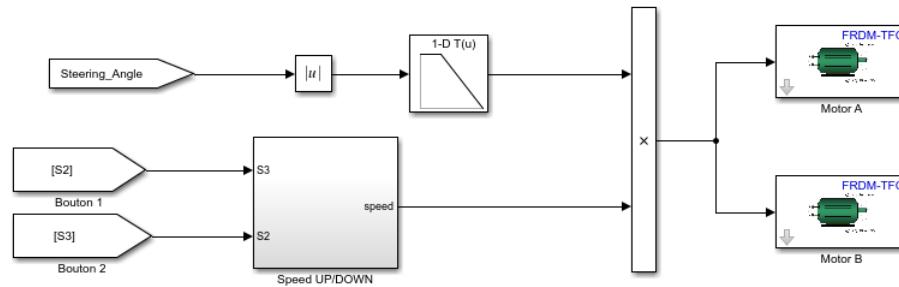


FIGURE 20 – Commande des moteurs à propulsion

9 Boutons Poussoirs

Afin de rendre notre programme plus flexible, nous nous sommes servi des boutons poussoirs, décrits précédemment, dans la carte des capteurs. Afin de détecter leurs états, nous avons utilisé le bloc suivant :

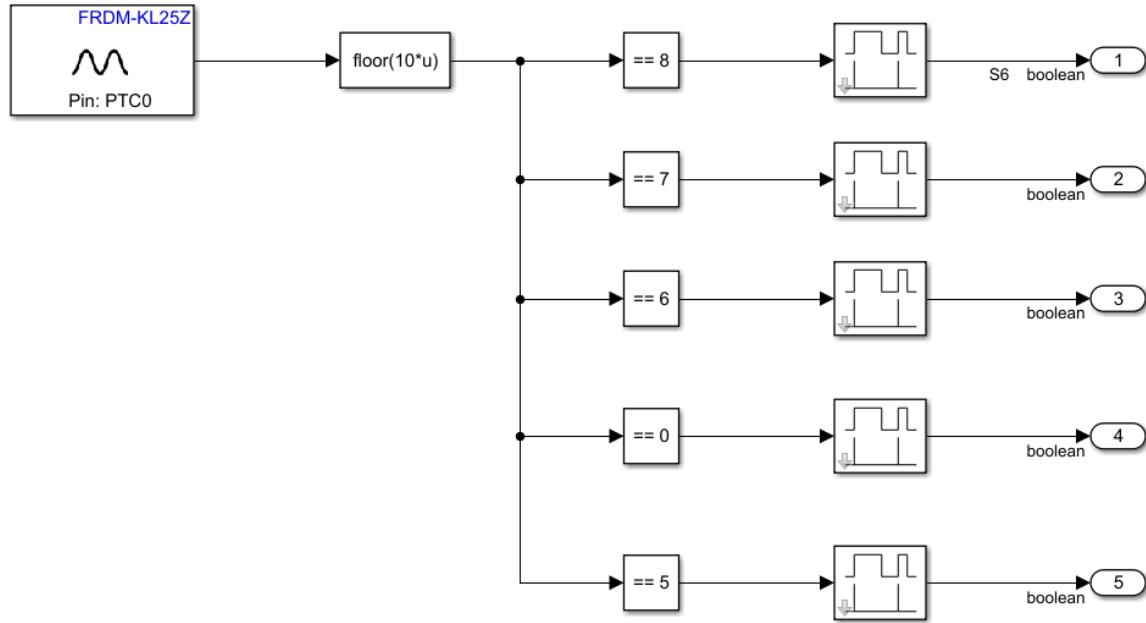


FIGURE 21 – Bloc des boutons poussoirs

En effet, les boutons poussoirs partagent, par le biais d'un circuit analogique, la même broche analogique PTC0, donc il suffit de lire la valeur de cette broche pour détecter lequel des boutons a été appuyé. De plus, vu que nous nous sommes intéressés que par l'appui et non pas par

la longueur d'appui, nous avons ajouté des détecteurs de crêtes dont la sortie est binaire et vaut 1 lorsqu'il y a un appui (front montant).

La répartition des boutons a été faite comme suit :

- Le bouton 1 permet de calibrer automatiquement la luminosité des pixels (cette partie est traité dans la suite du rapport).
- Les deux boutons 2 et 3 permettent respectivement d'augmenter et de diminuer l'offset du servo-moteur.
- Les deux boutons 4 et 5 permettent respectivement d'augmenter et de diminuer la vitesse des moteurs de propulsion.

10 Quelques autres améliorations sur les performances du véhicule

Dans cette partie, nous suggérons des diverses améliorations à apporter au véhicule pour améliorer ses performances.

10.1 Point mort de la direction (Dead Spot)

Nous avons remarqué que lorsqu'on augmente le gain de direction à une valeur élevée, le véhicule risque de se zigzaguer dans les deux sens, même s'il doit rouler tout droit. En effet, notre système de direction n'émet un signal de direction nul que lorsque la caméra renvoie un indice de 64 exactement. Si la caméra renvoie un 63 ou un 65, un signal de direction différent de zéro se produira, ce qui obligera le véhicule à se déplacer vers la droite ou la gauche. Pour résoudre ce problème, nous allons ajouter un point mort à la direction où le signal de direction sera nul sauf si le signal de direction est supérieur à un seuil. Nous pouvons facilement implémenter ce point mort avec Lookup Table comme montré dans la figure 22.

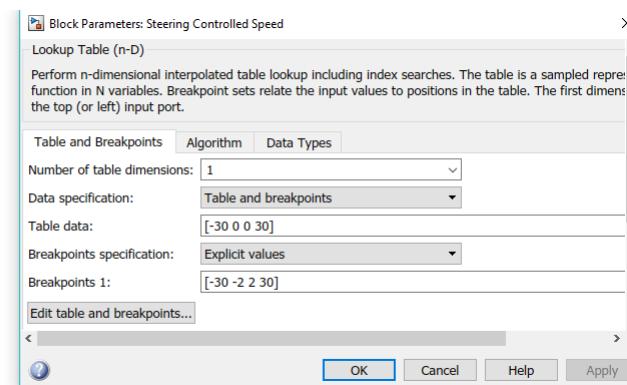


FIGURE 22 – Lookup table implémentant le point mort de la direction

10.2 Contrôle de la direction par les roues arrières

L'une des améliorations les plus importantes que nous puissions apporter est le contrôle de la direction par les roues arrières. Le principe est comme suit : Si nous appliquons le même couple à chacune des roues arrières, le véhicule aura tendance à rouler droit. Cependant, si nous

appliquons un couple maximal à la roue arrière droite et aucun couple au pneu arrière gauche, le véhicule voudra virer à gauche et vice versa. Nous pouvons utiliser cette propriété pour aider le véhicule à tourner avec moins d'angle de braquage et plus de contrôle.

On va se baser sur la valeur du signal de direction envoyé au servomoteur ainsi que son signe pour savoir dans quel sens le véhicule va tourner. Nous utilisons tout simplement des Lookup Tables pour définir les fonctions linéaires qui permettent de réduire/augmenter la vitesse d'un moteur en prenant le signal de direction comme argument. Le schéma fonctionnel est donné dans la figure 23

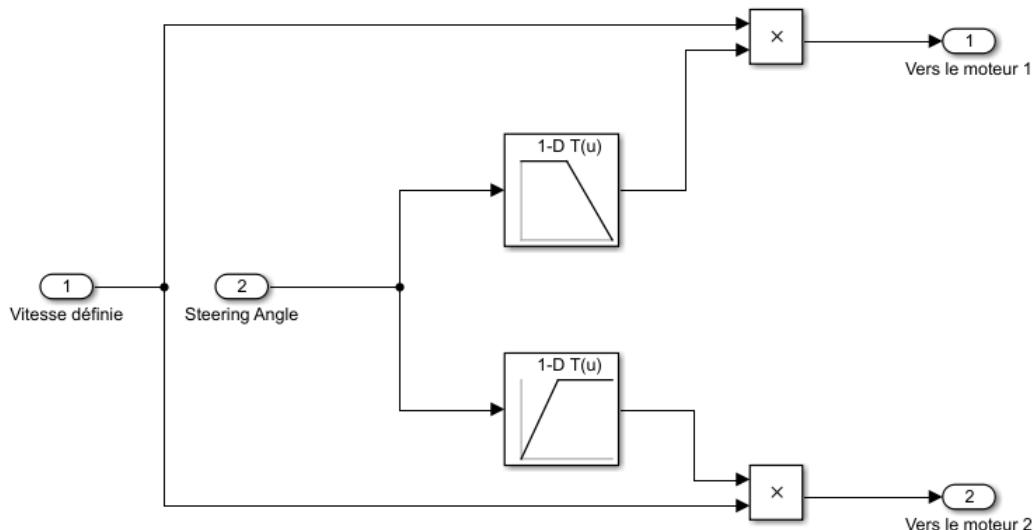


FIGURE 23 – Contrôle de la direction par les roues arrières

10.3 Calibrage de la luminosité de la caméra

En effet, les capteurs de la caméra ne sont pas parfaitement identiques, et son objectif ne permet pas à ces capteurs de percevoir la lumière par la même intensité même si la luminosité de la salle est homogène. Afin de remédier cette imperfection, nous avons fait un calibrage de la luminosité. En effet, avant de lancer la voiture, nous la mettons sur un fond blanc dont la luminosité est homogène et uniforme, puis nous faisons une acquisition par la caméra de la voiture. Ensuite, nous prenons le pixel du milieu comme référence et nous calculons le gain entre la luminosité des autres pixels par rapport à la luminosité de cette référence. Ainsi, lorsque la voiture roule sur la piste, à chaque acquisition de la caméra, nous multiplions chaque pixel par le gain correspondant.

Cela permet de faire des acquisitions crédible et améliore significativement les performances de la voiture et l'interprétation des virages.

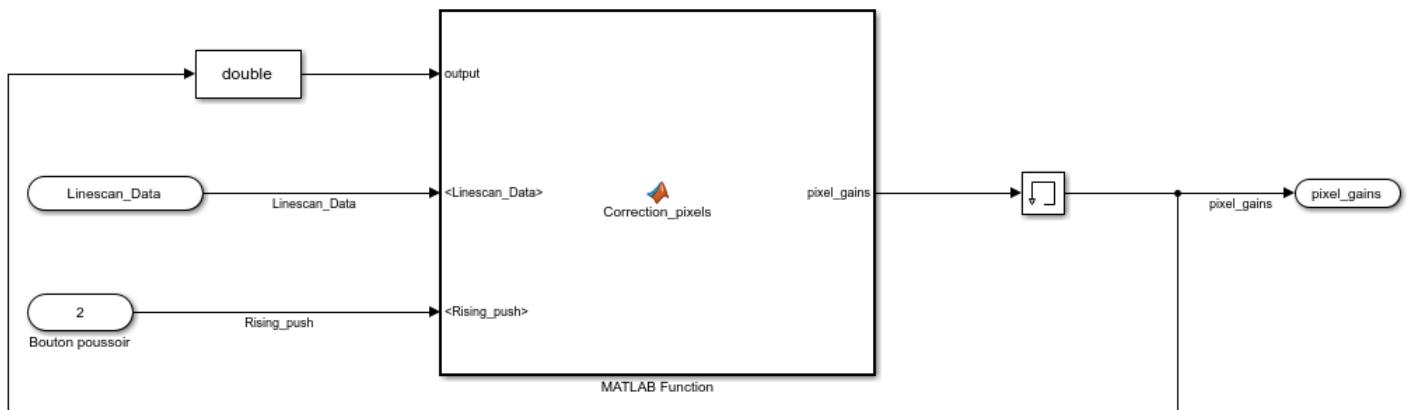


FIGURE 24 – Système du calibrage de la caméra

11 Modèle complet

L'étude réalisé tout au long de la période de préparation nous a amené à avoir une voiture autonome qui roule parfaitement sur notre circuit bouclé malgré quelques défauts lorsque la vitesse de la voiture est grande. Le schéma final de notre approche est présenté dans la figure 25

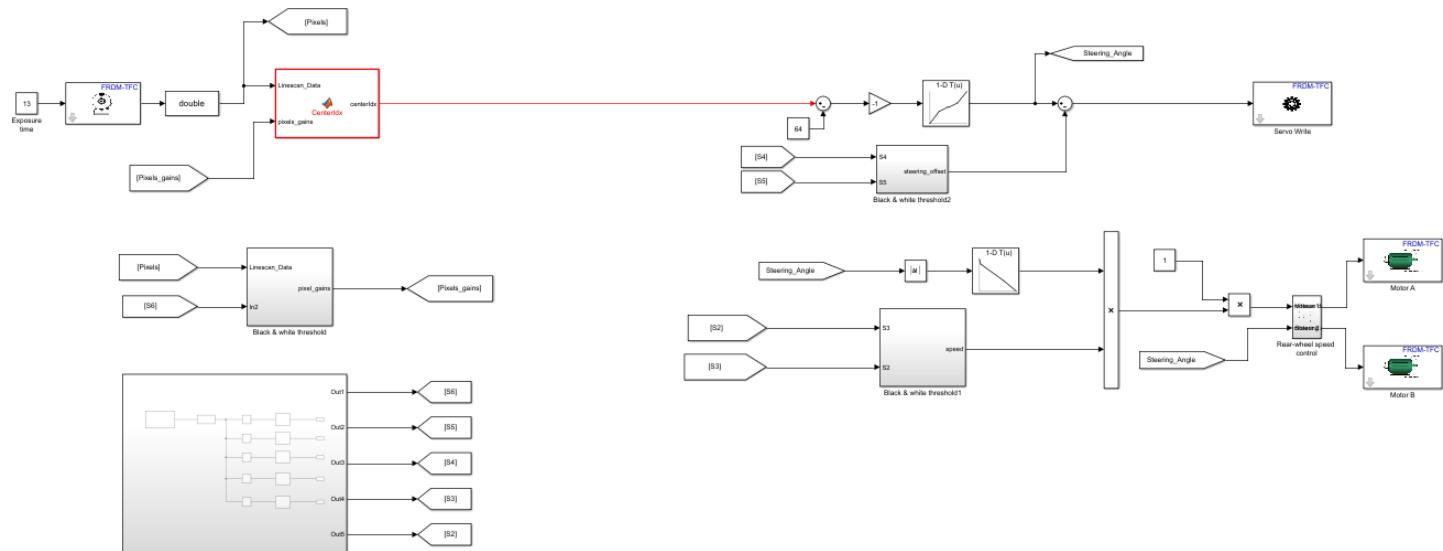


FIGURE 25 – schéma final de notre implémentation

12 Communication bluetooth

Au départ, on a commencé de chercher dans les bibliothèques de notre micro-contrôleur KL25Z afin de trouver celles qui permettent d'établir la communication par Bluetooth. Sur Simulink Matlab, nous sommes limités par les bibliothèques fournies par le package NxP qui ne contiennent pas des modules de communications autres que la liaison série. D'où le grand avantage de programmer tous en C.

Notre première approche consiste à utiliser un autre qui possède un module de communication série UART (asynchrone), ce dernier peut être utilisé comme protocole de communication avec un module bluetooth (HC-04 par exemple). En utilisant l'IDE de Mbed, vous pouvez utiliser la librairie "*MODSERIAL.h*", elle vous permet d'initialiser et de gérer la connexion bluetooth. En effet, les pin TX et RX du module bluetooth sont respectivement PTE22 et PTE23 utilisant le module UART2 du μ C. Un code exemple est joint parmi les références ci-dessous. La seule difficulté rencontrée est le fait de rassembler les deux micro-contrôleurs qui se manipulent sur deux interfaces différentes (un sur Matlab et l'autre sur IDE de Mbed).

Nous sommes partis sur l'idée de connecter la sortie du micro-contrôleur qui gère la connexion Bluetooth à une broche d'entrée analogique sur le KL25Z. Puis, on récupère cette information sur Matlab pour le traitement. Malheureusement, on n'avait pas des bons résultats, déjà à cause du fait que les valeurs transmises sur des broches analogiques ne peuvent pas dépasser Vdd et aussi pour bien traiter les informations reçues, un système qui détermine la réception de nouvelles données dans le passage au Matlab est indispensable.

Dans la suite, on va présenter les fonctionnalités permettant de gérer la communication Bluetooth en C (sur l'interface CodeWarrior) pour donner une base pour les prochains candidats, sachant qu'on n'avait pas la possibilité de les implémenter sur notre véhicule. Les codes sources sont fournis dans les références ci-dessous.

Un module Bluetooth HC-05 a été utilisé pour communiquer entre le MCU du véhicule et un ordinateur en se basant sur les fonctions du module **TFC_UART** du micro-contrôleur KL25Z. Ce module contient plusieurs fonctions pour transmettre et recevoir des données, comme *uart_putchar* pour placer un caractère sur le FIFO UART TX chaque fois que l'espace est disponible.

Pour pouvoir transférer des données de l'ordinateur au véhicule, on pourra utiliser Matlab. Les données seront de trois octets (un octet chacune) : un Bluetooth flag, un identifiant et la nouvelle valeur souhaitée. À partir de Matlab, l'envoi peut être programmé comme une machine d'états présentée dans la figure 26.

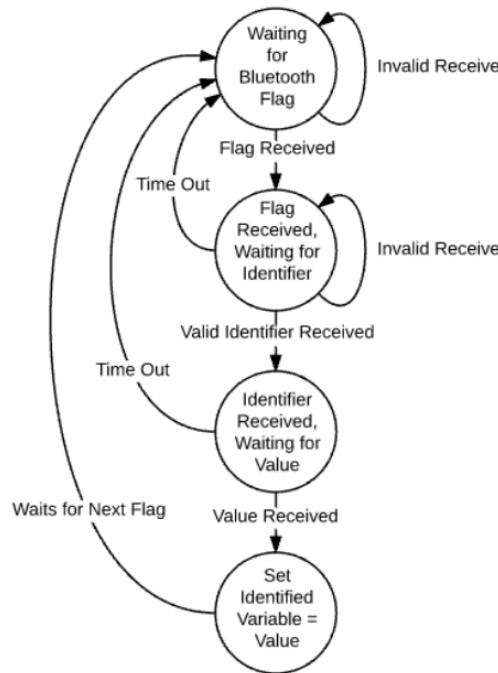


FIGURE 26 – Machine d'états de l'envoi des données par bluetooth

L'état initial consisterait à interroger le registre de données de réception pour UART2, l'UART Bluetooth, par le véhicule, afin de vérifier si le registre de réception de données était plein. Si tel est le cas, il renverra ces données en les répétant jusqu'à ce que l'indicateur de réception Bluetooth soit reçu. Lorsque l'indicateur Bluetooth a été bien reçu, il passe à l'état suivant où il attendra l'octet suivant, ce qui identifiera le type de données qui sera reçu dans l'octet suivant, à savoir une nouvelle valeur de vitesse des moteurs, un nouveau temps d'acquisition de la caméra, etc. Le troisième octet serait la valeur numérique de cette donnée. Après avoir reçu la valeur, l'identifiant sera utilisé dans une instruction *switch* pour déterminer la fonction à appeler, avec la valeur transmise en tant qu'argument.

Cette première approche peut être efficace dans l'envoi des valeurs. Mais, au niveau de la réception comme elle utilise utilise la scrutation des registres, ce qui prendrait du temps de traitement.

Afin d'optimiser le temps de traitement, on pourra programmer les fonctionnalités d'envoi et réception vues sur Matlab directement en C en se basant sur les interruptions pour UART.

L'implémentation d'une communication Bluetooth permet d'avoir plusieurs fonctionnalités pour le débogage, telles que la modification de la vitesse pendant le fonctionnement du véhicule, l'arrêt du véhicule à distance, l'activation et la désactivation d'autres fonctions, etc.

13 Conclusion

Il est indéniable que la NXP CUP nous a permis d'avoir un contact concret avec le monde de la programmation embarquée. En effet, la commande autonome de la voiture nécessite une étude minutieuse pluridisciplinaire englobant le traitement d'images, pilotage d'informations circulant au travers du μ C, et une étude de la dynamique du servo-moteur et les moteurs à propulsions. Nous serons ravi de revivre cette aventure avec les nouveaux candidats en les aidant avec l'expérience que nous avons acquises.

14 Références :

<https://nxp.gitbook.io/nxp-cup-hardware-reference-alamak/> Tutoriel proposé par NXP semiconductor

https://github.com/amjadameur/nxpCup_git.git notre répertoire github

https://os.mbed.com/users/GerritPathuis/code/HC05_KL25Z_Hello_PC/ Module Bluetooth sur Mbed