Rapport final : NXP Cup

Projet R&D

Par

Rio Guillaume

et

Bhikir Oumnia

Robotique et système autonomes

Polytech Nice-Sophia

Université Côte d'Azur

Date : 20 Janvier, 2025

Sommaire

Introduction

La robotique est un monde riche, en constante évolution, offrant une multitude d'opportunités pour apprendre, innover et évoluer. C’est dans cet esprit que la NXP Cup, se positionne, permettant aux étudiants de développer un véhicule autonome capable de naviguer sur une piste dévoilée le jour de la compétition.

Ce rapport a été conçu non seulement pour documenter le travail réalisé dans le cadre de ce projet, mais surtout pour servir de ressource aux futurs étudiants en 3e et 4e année de robotique qui reprendront notre projet. Notre objectif est de vous fournir une ressource qui vous accompagnera pas à pas, de la configuration initiale au premier tour de piste

Dans ce projet, nous avons utilisé deux types de caméras, une ArduCam et une linescan caméra, qui nécessite la compréhension de concept tel que la communication SPI ou encore l’estimation de la position via un filtre Kalman.

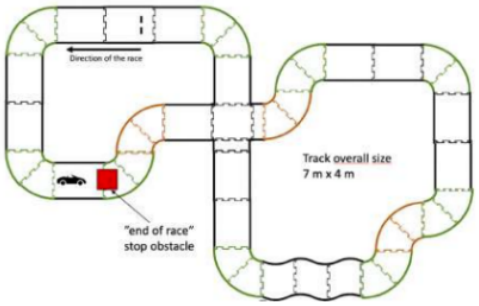
En suivant ce guide, nous espérons que vous serez non seulement en mesure de reproduire ce que nous avons réalisé, mais d'aller au-delà, en apportant vos propres améliorations. Que la compétition commence pour vous, futurs ingénieurs en robotique !

1. NXP Cup

La NXP Cup est une compétition étudiante internationale organisée par NXP Semiconductors. Elle vise à encourager les étudiants à développer des compétences en technologie, ingénierie et programmation en concevant et programmant une voiture autonome. Les participants doivent construire un véhicule capable de naviguer sur un circuit inconnu, délimité par des lignes noires, de manière totalement autonome.

C'est une course de vitesse, où la voiture doit compléter le circuit le plus rapidement possible.Et une épreuve de précision, où la voiture doit s'arrêter à moins de 10 cm d'un obstacle sur la piste après un tour rapide.

Les voitures doivent être construites avec des composants NXP et programmées exclusivement avec l'environnement MCUXpresso.



2. Matériels et Logiciel Requis

Pour mener à bien le projet, cette section détaillera les outils nécessaires pour reproduire, comprendre et étendre notre travail.

2.1 Liste du Matériel

- Carte de Développement NXP FRDM K22F avec processeur ARM Cortex M4 et compatible avec l'environnement de développement MCUXpresso.
- Caméra :
  LineScan Camera (pour une détection précise des lignes grâce à sa capacité à capturer une dimension unique des données)
  ArduCam- SPI, utilisé ultérieurement pour améliorer notre vue sur la piste
- Moteur :
  Deux moteurs Brushless a2212/15t 930kv pour la propulsion, contrôlés avec deux ESC blheli_32.

- Un Servomoteur futaba s3010, pour gérer la rotation de la voiture
- Un capteur de distance, la course se déroule sur deux temps, lors du deuxième tour de piste, les voitures doivent s'arrêter à une distance prédéfinie de 10 cm devant un obstacle, placé sur la piste par les juges
- Batterie et régulateurs de tension: nous possédons une batterie Lipo 5000mAh, 11.1V 3C1P 55.5Wh. Nous utilisons un régulateur de tension (Power module V6.0) afin de générer une tension de 7V pour les moteurs brushless et une tension de 5V pour alimenter la carte. Un autre régulateur de tension (hobbywing 3a ubec) uniquement pour alimenter le Servomoteur qui nécessite 3A.
- Structure du véhicule, châssis fournis par NXP avec guide pour montage
- Câbles et Connecteurs

2.2 Liste des Logiciels

- MCUXPresso : Environnement de développement intégré pour la programmation de la FRDM K22F, permet de compiler, déboguer et tester les applications.

3. Prise en main de MCUXPresso

MCUXpresso est l'environnement de développement intégré recommandé pour programmer et déboguer les cartes NXP, y compris la FRDM utilisée dans ce projet. Cette section comment prendre en main cet outil.

- Installation MCUXpresso : sur le site officiel de NXP, télécharger la dernière version de MCUXpresso IDE, en suivant les instructions d'installation adapté au système d'exploitation utilisé. (lien en annexe)
- Importance du SDK dans MCUXPresso : Un SDK est un kit de développement logiciel. Il contient un ensemble d'outils de création spécifiques à notre carte. Durant la programmation vous aurez besoin de composants tels que des débogueurs, des compilateurs, des drivers et des bibliothèques contenant des exemples, tout ceci est contenu dans le SDK.
- Télécharger le SDK pour la FRDM-K22F : Il vous est donc impératif de télécharger le SDK correspondant à votre carte afin de pouvoir créer et exécuter un programme sur votre carte. (explication de la procédure et lien en annexe)
- Création projet : Si vous souhaitez recommencer un projet vous pouvez aller dans le menu principal, cliquez sur New Projet, sélectionnez la carte FRDM K22F et configurez un projet vide ou existant.

4. 1 Connexion de la Linescan Camera via ADC

La parallax TSL1401-DB est une caméra linéaire capable de capturer des informations en une seule dimension. Elle utilise une broche analogique (A0) pour transmettre les données et des broches numériques pour les signaux de synchronisation.

Description des connexions :

- GND : connecté à la masse de la carte
- Vdd : alimentation de la caméra ((3.3V Arduino, 5V NXP)
- A0 : sortie analogique connectée à l'entrée ADC de la carte
- SI : entrée numérique pour la synchronisation, GPIO configuré en sortie
- CLK : entrée numérique pour l'horloge, GPIO configuré en sortie

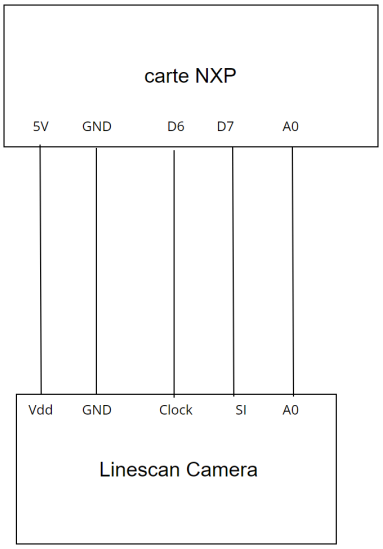


Figure X: Configuration pin Linescan

Cycle de fonctionnement :

- La broche SI démarre un nouveau cycle en envoyant une impulsion
- Le signal CLK permet de synchroniser la lecture de chaque pixel un à un
- Les données analogiques pour chaque pixel sont envoyé vias l'ADC à chaque bord descendant de CLK

4.2 Connexion de l'ArduCam via SPI

La caméra ArduCAM est utilisée pour compléter la linescan en offrant plus de flexibilité et une meilleure visibilité . Elle communique via le protocole SPI, qui assure une transmission plus rapide et fiable des données.

Description des connexions :

- MISO, MOSI, SCK, CS : connectés respectivement aux broches SPI de la carte.
- GND : connecté à la masse
- VCC : alimentation de la caméra en 5V

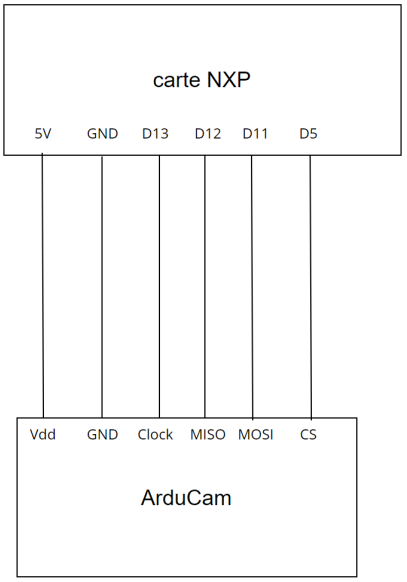


Figure X: Configuration pin ArduCam

Cycle de fonctionnement :

- La carte envoie une commande SPI pour configurer ou récupérer des données de l'ARduCAM
- Les données sont échangées en utilisant les broches MOSI (transmission) et MISO (réception)

Malgré de nombreux essais, l'intégration de l'ArduCAM via SPI n'a pas abouti. Les problèmes rencontrés comme des **BusFault**, **HardFault**, des données corrompues ou une absence de retour, n'ont pu être résolus, même après avoir désactivé le **DMA**, ajusté les timings SPI et vérifié les connexions. Ces difficultés, combinées à une documentation limitée, ont montré que, dans ce contexte, la **linescan camera** reste une solution plus simple et fiable. Je conseille aux prochains étudiants de ne pas trop s'en intéresser, à moins de disposer des ressources nécessaires pour approfondir l'ArduCAM.

4.3 Connexion du capteur à ultrason hc-sr04

Le capteur à ultrason est utilisé pour détecter s'il y a un obstacle sur la piste. Pour ce faire on utilise deux pin, le pin Trigger afin de notifier le capteur qu'il doit réaliser une mesure et le pin echo pour déterminer la distance.

Cycle de fonctionnement :

Lorsque le pin trigger est à l'état haut, le capteur envoie 8 impulsions à 40Khz et place le pin Echo à l'état haut. Lors de la réception d'une des impulsions, le capteur passe l'état du pin echo à bas. Ainsi en mesurant le temps que le pin echo est resté à haut, on peut déterminer la distance.

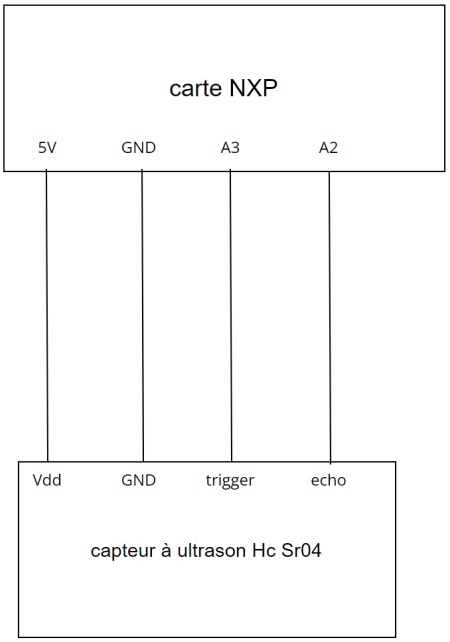


Figure X: Configuration pin ArduCam

5. Traitement d'image et détection de la piste

Le traitement d'image et la détection de la piste sont au cœur de ce projet, ils permettent au véhicule de naviguer sur le circuit. Cette section détaille les étapes pour analyser les données de la linescan camera, détecter les bords de la piste.

- ***Structure des Données de la Linescan Camera***

La linescan camera fournit une séquence de 128 pixels, chaque pixel représentant une intensité lumineuse capturée. Les valeurs analogiques varient généralement de 0 (noir) à 4095 (blanc). Cette information permet de repérer les zones blanches (piste) et noires (bordures) sur le circuit.

- ***Détection des Bordures***

Les valeurs analogiques sont lues via l'ADC et stockées dans un tableau, un filtre exponentiel est appliqué pour réduire le bruit et lisser les données. On parcoure le tableau pour récupérer les transistions noir-blanc (bordure gauche) et blanc-noir (bordure droite).Les indices de ces transitions sont enregistrés pour identifier les bords de la piste.

- ***Calcule de la trajectoire :***

Le centre de la piste est calculé comme moyenne des indices des bordures détectées : $centre = (leftEdge + rightEdge) / 2$

L' erreur de position est ensuite définie comme la différence entre le centre de la caméra (64 pour 128 pixels) et le centre de la piste détectée.

- ***Direction à suivre :***

- Si l'erreur est positive, le véhicule doit tourner à gauche.
- Si l'erreur est négative, le véhicule doit tourner à droite.
- Gestion des cas particuliers :
    - Une seule bordure détectée : Si seulement le bord gauche est détecté, le véhicule est trop à gauche, corrigez en tournant à droite. Pareillement pour le bord droit.
    - Aucune bordure détectée : Ce cas indique une sortie de piste. Pour y remédier, on réduit la vitesse pour revenir à la piste. Si le problème persiste, on réajute les seuils de transition noir-blanc et blanc-noir. Il va de même pour les erreurs en cas de forte lumière ambiante.
    - Si le signal est instable ou bruité : on vérifie la calibration de la caméra et les branchements avec un oscilloscope. Un filtre a été ajouté dans le but de lisser le signal.

- ***Implémentation de la caméra***

Le code pour la caméra comprend plusieurs fonctions essentielles : **clockPulse(), delay(), initADC(), readADC(), readFilteredADC(), detectTrack() et adjustDirection().**

La fonction **clockPulse()** envoie une impulsion d'horloge à la caméra. Elle règle la broche d'horloge à l'état haut pendant 1 microseconde, puis à l'état bas pour la même durée. Cette impulsion est nécessaire pour synchroniser la lecture des pixels avec le capteur de la caméra.

La fonction **delay(**) introduit un délai en exécutant une instruction NOP (No Operation) un certain nombre de fois. Cela permet de contrôler le timing des opérations cruciales pour la synchronisation avec la caméra.

La fonction **initADC() i**nitialise le convertisseur analogique-numérique (ADC) sur la carte. Elle configure l'ADC avec les paramètres par défaut, exécute une calibration automatique et prépare la structure de configuration du canal pour la lecture des valeurs analogiques.

La fonction **readADC()** lit une valeur analogique à partir du capteur de la caméra via l'ADC. Elle configure le canal de l'ADC, attend que la conversion soit terminée, puis retourne la valeur convertie, nous avons essayé d'améliorer cette fonction en introduisant la fonction **readFilteredADC()** qui applique un filtrage à la valeur lue par l'ADC pour lisser les variations. Elle utilise un filtre exponentiel avec un facteur ALPHA pour combiner la nouvelle lecture avec la précédente, réduisant ainsi le bruit.

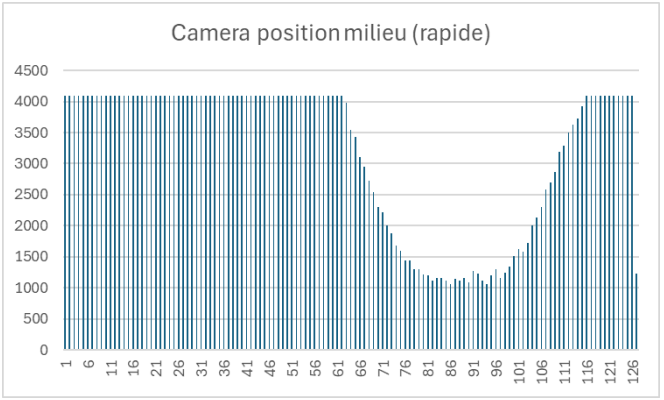
La fonction **detectTrack()** analyse les valeurs des pixels pour détecter les bords de la piste. Elle recherche deux séquences de pixels noirs séparées par une distance minimale, représentant les bords gauche et droit de la piste. Si elle ne trouve qu'une seule ligne, elle détermine si le robot est trop à gauche ou à droite.

La fonction **adjustDirection()** ajuste la direction du robot en fonction de la position détectée de la piste. Elle calcule la déviation par rapport au centre de l'image et imprime la direction à suivre.

En résumé, notre implémentation initialise et configure l'ADC, lit les valeurs des pixels de la caméra, applique un filtrage pour lisser les lectures, détecte les bords de la piste et ajuste la direction du robot en conséquence. Ces fonctions travaillent ensemble pour permettre au robot de suivre la piste de manière autonome en utilisant les données de la caméra

Résultats observés :

La linescan caméra ne renvoie pas d'image visuelle, elle capture des images lignes par lignes plutôt que par matrice de pixels comme les caméras traditionnelles, chaque pixel de cette ligne capture la lumière en petite portion, et l'horloge contrôle la fréquence à laquelle le pixel est lu. Ainsi il est impossible de visualiser ce que voit la caméra. Nous avons donc décidé d'afficher la valeur de l'intensité allant de 0 pour le noir à 4095 valeur pour le blanc pur. Ces valeurs lues séquentiellement à chaque cycle d'horloge sont ensuite traitées pour former une image complète ligne par ligne.



6. Contrôle moteurs et navigation

La capacité de la voiture à se déplacer de manière fluide et à ajuster sa direction en fonction des données capturées repose sur le contrôle des moteurs. Dans cette section, nous détaillons les étapes nécessaires pour configurer et piloter les moteurs brushless et le servomoteur.

1. Configuration des Signaux PWM des moteurs Brushless :

Les moteurs brushless contrôlent les roues arrière et fournissent la propulsion nécessaire. Leur vitesse est régulée par des signaux PWM envoyés via les ESC.
Chaque moteur est contrôlé par un signal PWM (Fréquence : 48 kHz) généré par les FTM de la carte NXP, les broches utilisées sont le suivantes :

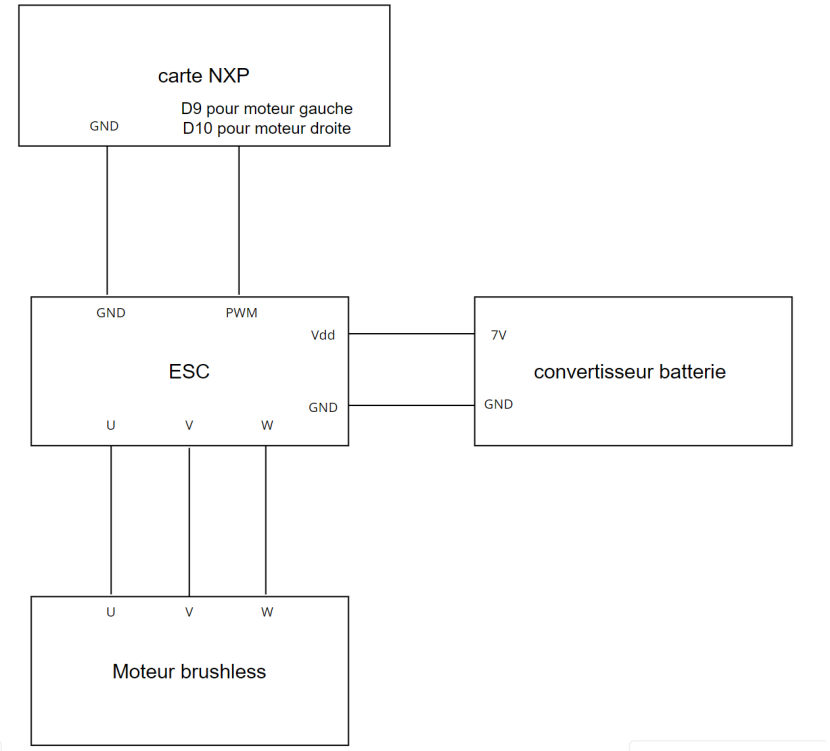


Figure X: Configuration pin moteur brushless

Moteur gauche : FTM0 Channel 6 (Pin D9).
Moteur droit : FTM0 Channel 4 (Pin D10).

2. *Implémentation des moteur Brushless*

Afin de faire fonctionner nos moteurs brushless, il nous suffit de générer un signal PWM. Pour ce faire nous utilisons des channels FTM (FlexTimerModule) que nous configurons à l'aide des fonctions mise à disposition dans le SDK. Afin de choisir la fréquence que l'on va utiliser pour notre PWM nous sommes allés dans la documentation de l'ESC et la fréquence maximale qu'il supporte est de 48kHz. Nous ne possédons pas de contrainte spécifique, nous avons donc décidé de les contrôler à cette fréquence.
Ensuite, il fallait déterminer la range de fonctionnement de nos moteurs (le dutycycle du PWM). De manière expérimental nous avons déterminé que la range de fonctionnement était entre 30% et 65% du dutycycle.
De plus, nous avons configuré nos esc afin que nos moteurs puissent avancer et reculer. Ainsi si le dutycylce est compris entre 30 et 46 nos moteurs recule, entre 48 et 65 nos moteurs avance et pour 47 nos moteurs sont arrêtés.

3. *Configuration du Signal PWM du servomoteur:*

Le servomoteur contrôle la direction en ajustant les roues avant. Il est également commandé par un signal PWM (Fréquence : 50Hz).

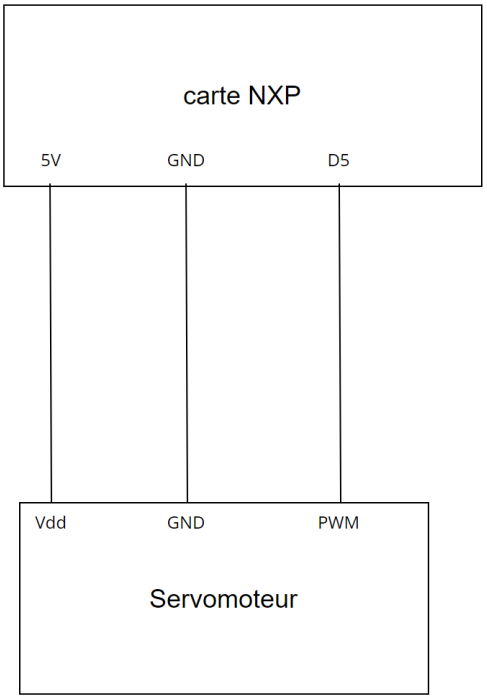


Figure X: Configuration pin servomoteur

Canal utilisé : FTM2 Channel 0 (Pin D5).

4. Implémentation du servo moteur:

Pour le servomoteur nous avons suivi la même méthodologie que pour les moteurs brushless. Afin de déterminer la fréquence de fonctionnement nous sommes allés voir la datasheet du servomoteur et nous avons trouvé que pour le contrôler il faut un signal PWM à 50Hz. Nous avons donc initialisé un autre channel à cette fréquence. Enfin nous avons déterminé le dutycycle expérimentalement car l'angle de rotation est limité mécaniquement il ne faut donc pas envoyer un signal qui pourrait faire forcer le servomoteur et le casser.

7. Algorithme

Dans cette section, nous nous concentrerons sur la logique de notre code, qui est structuré en deux parties distinctes.

- La première partie correspond au code procédural. Elle contient le traitement de la caméra, lequel détermine les actions à effectuer sur les moteurs.
- La seconde partie repose sur un code basé sur des interruptions. Cette approche permet de gérer le fonctionnement du capteur de distance sans bloquer l'exécution de la première partie.

Afin de synchroniser notre carte avec nos capteurs, il est essentiel de disposer d'un moyen permettant de mesurer le temps avec précision. Pour cela, nous utilisons une interruption système appelée Systick_Handler. Cette interruption repose sur l'horloge système fonctionnant à une fréquence de 120 MHz. Elle peut être configurée de manière à se déclencher après un nombre précis de ticks (SysTick_Config). Nous avons ainsi défini que cette interruption se déclenchera toutes les 1 µs.

Grâce à ce timer, nous pouvons mesurer les distances et transmettre les trames nécessaires à la caméra pour recevoir les informations.

À partir des données fournies par la caméra, nous pouvons déterminer le comportement de notre contrôleur. Initialement, nous avons envisagé d'utiliser un contrôleur PID, pensant qu'il serait le plus adapté. Cependant, l'angle de vue restreint de notre caméra ne permet pas de visualiser simultanément les deux lignes, ce qui limite son utilité. Avec une seule information à traiter, un PID n'est pas nécessaire.

Nous avons donc opté pour une logique plus simple : si une ligne est détectée à droite, la voiture tourne à gauche, et inversement.

Amélioration apportées :

Au cours du développement du projet, plusieurs améliorations ont été mises en œuvre pour optimiser les performances :

1. Ajout d'une ArduCAM pour Complémenter la Linescan Camera :

- Initialement, seule la linescan camera était utilisée pour détecter les bordures de la piste. Cependant, pour explorer des fonctionnalités avancées et bénéficier d'une plus grande flexibilité, une **ArduCAM** a été intégrée au projet.
- Bien que cette intégration n'ait pas abouti à des résultats exploitables, elle a permis d'approfondir les connaissances sur la communication SPI et d'envisager des solutions évolutives pour de futurs projets.

2. Calibration et Filtrage des Données de la Linescan Camera :

Une attention particulière a été portée à la calibration de la caméra en ajustant la mise au point et l'exposition manuelle. Aussi, un filtre exponentiel a été ajouté pour réduire le bruit des données capturées, ce qui a considérablement amélioré la précision de la détection des bordures.

3. Optimisation de la Navigation :

Les algorithmes de contrôle des moteurs et du servomoteur ont été ajustés pour fournir une réponse plus fluide et réactive, particulièrement dans les virages.La gestion dynamique de la

vitesse en fonction de l'erreur de position a permis de réduire les risques de sortie de piste tout en maintenant des performances élevées.

4. Renforcement des Connexions Matérielles :

Les câbles et connecteurs fragiles, notamment ceux de la linescan camera, ont été sécurisés avec des gaines pour éviter des interruptions dues à des déconnexions accidentelles.

Problème rencontrés :

Malgré les améliorations apportées, certaines problématiques ont persisté, preuve de contraintes de temps et limites techniques :

1. Problèmes avec la Communication SPI (ArduCAM) :

La caméra ArduCAM n'a pas pu être exploitée, malgré de nombreux tests et ajustements.Des erreurs fréquentes, comme des **BusFault** et **HardFault**, ainsi que des données corrompues, ont ralenti le processus. La documentation limitée et la complexité du protocole SPI ont ajouté à la difficulté.

2. Sensibilité de la Linescan Camera :

La linescan camera s'est montrée très sensible aux variations de lumière ambiante, nécessitant des ajustements fréquents de la mise au point et de l'exposition. Ces limitations ont compliqué les tests en conditions réelles, notamment sur des circuits avec des éclairages changeants.

3. Calibration des Moteurs :

Les deux ESC des moteurs brushless avaient des configurations initiales différentes, ce qui a nécessité une calibration manuelle pour harmoniser leur comportement. Le manque de documentation sur les ESC a rendu cette étape plus complexe.

4. Problèmes de Synchronisation :

La synchronisation des signaux pour les différents capteurs, notamment la linescan camera et le capteur ultrason, a parfois causé des conflits, nécessitant des ajustements dans le code.

4. Gestion du Temps et des Ressources :

Avec un délais serré, certaines fonctionnalités prévues, comme l'intégration complète de l'ArduCAM ou l'autocalibration de la linescan camera, n'ont pas pu être finalisées.

Leçons Tirées et Suggestions pour le Futur

Les défis rencontrés tout au long de ce projet ont constitué une expérience précieuse en matière de résolution de problèmes et de gestion des priorités. Pour les futurs étudiants qui poursuivront notre travail, il est conseillé de se concentrer en priorité sur l'optimisation de la

linescan camera, qui s'est révélée être une solution fiable malgré sa sensibilité aux conditions de lumière. Il serait également bénéfique d'explorer des alternatives matérielles pour la caméra, en privilégiant des options disposant d'une documentation claire et détaillée. Par ailleurs, il est essentiel de prévoir un temps dédié à la calibration des ESC et des capteurs, en documentant chaque étape pour simplifier le travail des prochaines équipes. Enfin, tester le véhicule dans des environnements variés, hors la salle dédiée au projet permettra d'anticiper les éventuels problèmes liés à la lumière ambiante ou à des surfaces irrégulières.

Conclusion

Ces deux années de travail sur la NXP Cup ont été bien plus qu'un simple projet académique : elles ont été une aventure technique. De la prise en main des premiers composants à la construction d'un véhicule autonome capable de naviguer sur un circuit inconnu, ce projet nous a permis de transformer des idées en solutions concrètes. Nous avons non seulement approfondi nos compétences en robotique et en programmation, mais également appris à faire face à des imprévus

Chaque étape a apporté son lot de défis, mais aussi de satisfactions. Qu'il s'agisse de calibrer les moteurs, d'optimiser la détection des lignes avec la linescan camera ou de tenter, parfois sans succès, d'intégrer de nouvelles technologies comme l'ArduCAM, chaque difficulté surmontée a enrichi notre expérience et renforcé notre détermination.

Ce projet nous a également appris l'importance de l'adaptabilité : lorsque certaines approches ne fonctionnaient pas, nous avons su rebondir et trouver des alternatives pour avancer.

À vous, les étudiants qui reprendrez ce projet, sachez que vous avez entre les mains une opportunité incroyable de développer vos compétences tout en vous amusant. Ce rapport transmet non seulement nos méthodes et solutions, mais aussi nos erreurs et nos leçons apprises, afin que vous puissiez aller encore plus loin.

Prenez le temps de comprendre chaque étape, expérimentez, et surtout, ne craignez pas l'échec – il fait partie du processus d'apprentissage.

Nous espérons que notre travail vous inspirera et que vous continuerez à porter ce projet avec autant d'enthousiasme et de passion que nous. Bonne chance, et que votre aventure dans la NXP Cup soit aussi enrichissante que la nôtre l'a été. **Ne lâchez rien, et surtout, amusez-vous !**

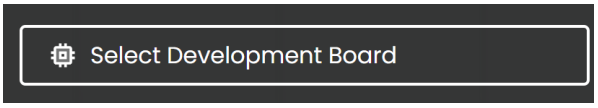
ANNEXE

Lien utile:

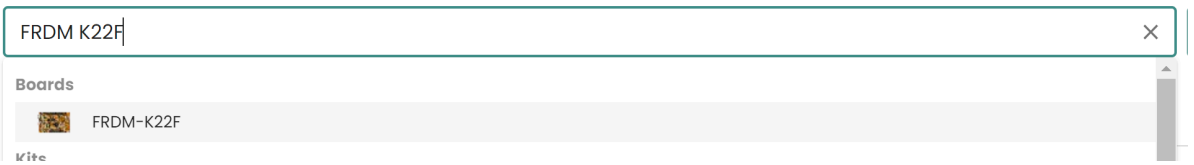
- datasheet esc BLHeli_32: https://img.banggood.com/file/products/20171201012630BLHeli_32%20manual%20ARM%20Rev32.x.pdf
- datasheet Brushless Moteur – A2212/15T – 930KV: https://bc-robotics.com/shop/brushless-motor-a221215t-930kv/
- datasheet Servo Moteur https://futabausa.com/wp-content/uploads/2018/09/S3010.pdf

Téléchargement:

- MCUxpresso https://www.nxp.com/design/design-center/software/development-software/mcuxpresso-software-and-tools-/mcuxpresso-integrated-development-environment-ide:MCUXpresso-IDE
- SDK pour la carte NXP FRDM K22F
    - Se rendre sur cette page https://mcuxpresso.nxp.com/en
    - Cliquer sur le bouton select Development Board

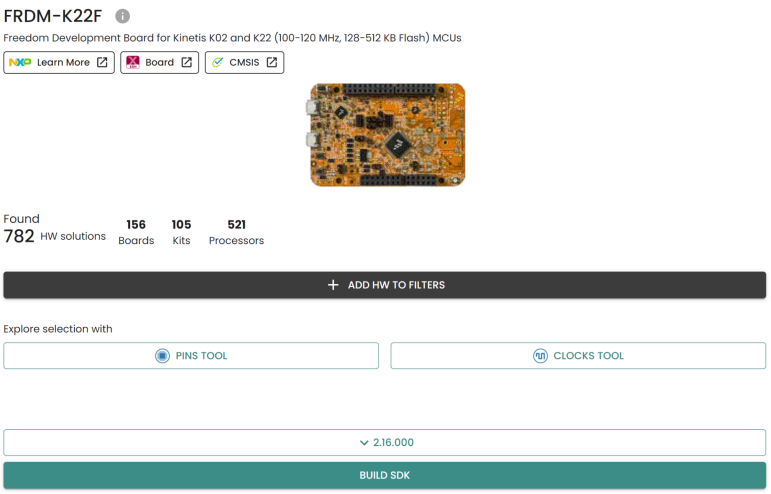


    - Se connecter à NXP (créer un compte si vous n'en possédez pas)
    - Dans la barre de recherche entré FRDM K22F



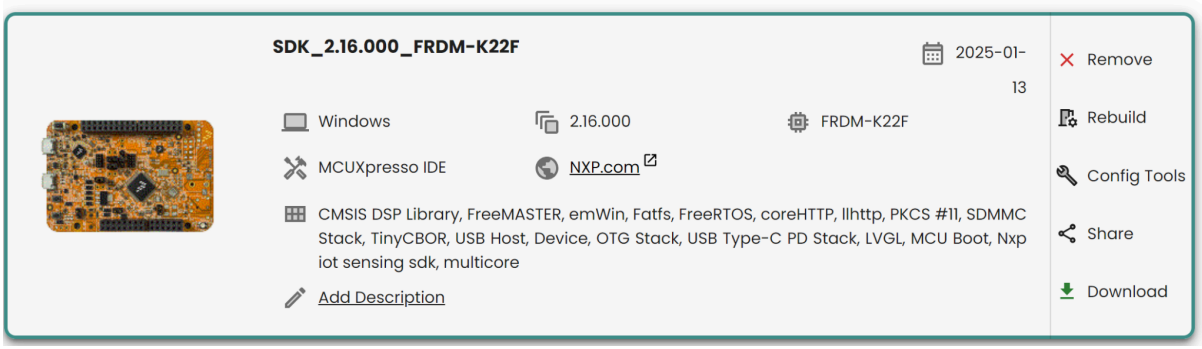
    - Cliquer sur la carte dans l'onglet Board

Cette page devrait se trouver en dessous (il faut scroller vers le bas)



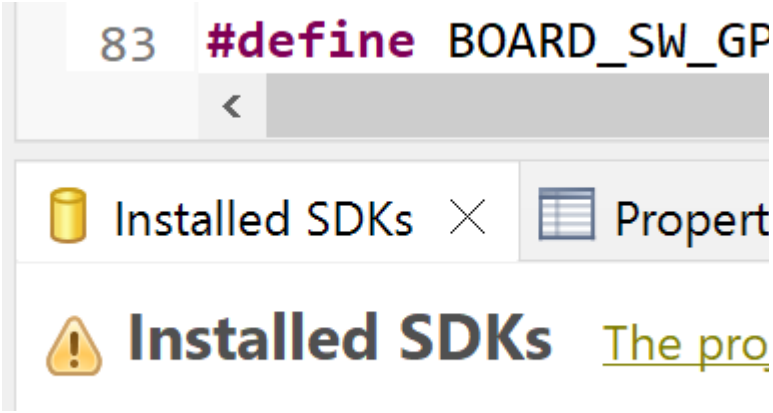
    - Vous pouvez cliquer sur build SDK,

- Choisissez vos paramètres en cochant les différentes options et en définissant votre OS (nous avons coché toutes les cases pour ne pas nous prendre la tête).
- Vous obtenez une page avec différentes archives, vous pouvez cliquer sur le bouton download en bas à droite

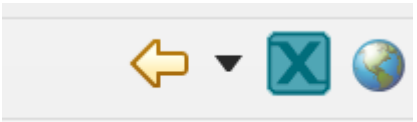


Pour importer le SDK dans MCUxpresso

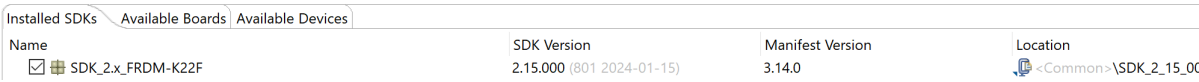
- en bas choisir l'onglet Installed SDKs



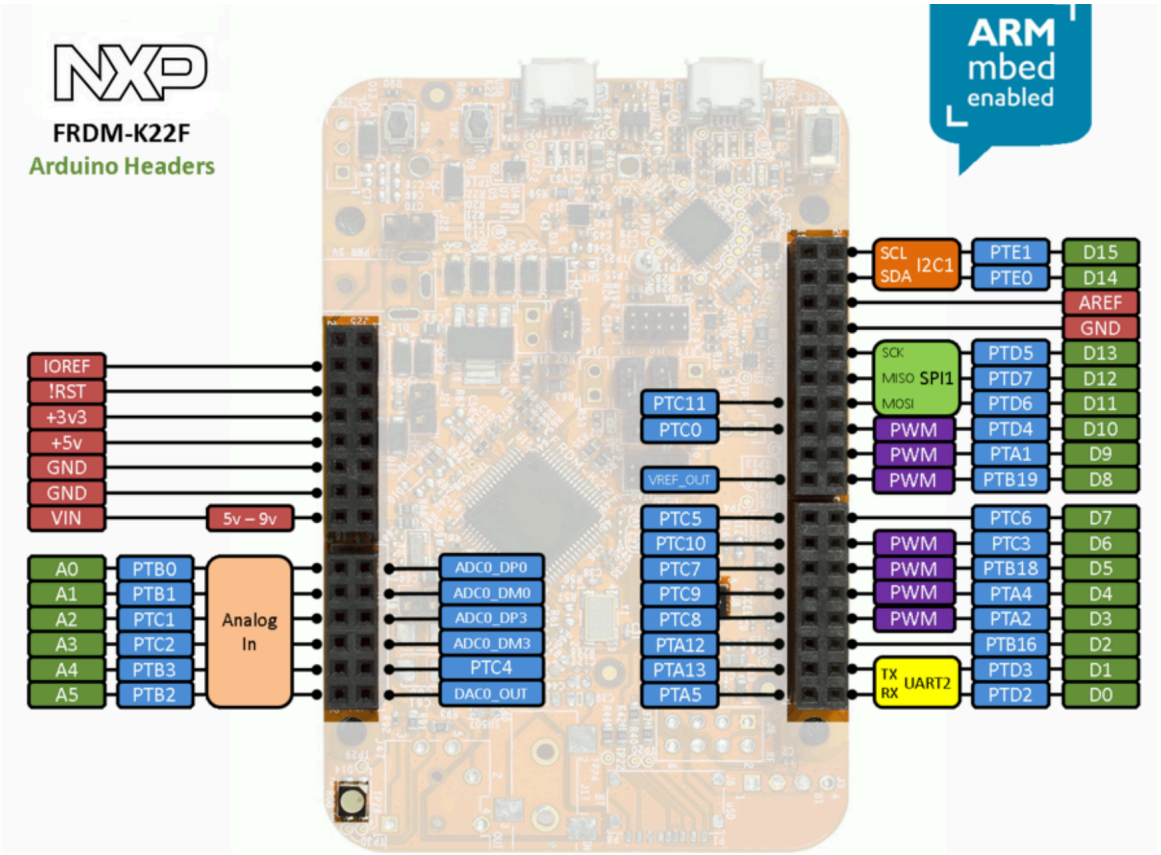
- puis cliquer sur la petite flèche à droite pour importer votre archive ou cliquer déposer votre archive directement dans l'onglet installed SDK



- Vous devriez obtenir quelque chose comme ça



Description des pins de la carte:


NXP
FRDM-K22F
Arduino Headers
ARM
mbed
enabled
IOREF
!RST
+3v3
+5v
GND
GND
VIN
5v – 9v
A0
A1
A2
A3
A4
A5
PTB0
PTB1
PTC1
PTC2
PTB3
PTB2
Analog
In
ADC0_DP0
ADC0_DM0
ADC0_DP3
ADC0_DM3
PTC4
DAC0_OUT
PTC11
PTC0
VREF_OUT
PTC5
PTC10
PTC7
PTC9
PTC8
PTA12
PTA13
PTA5
SCL
SDA
I2C1
SCK
MISO
MOSI
SPI1
PWM
PWM
PWM
PWM
PWM
PWM
PWM
TX
RX
UART2
PTE1
PTE0
PTD5
PTD7
PTD6
PTD4
PTA1
PTB19
PTC6
PTC3
PTB18
PTA4
PTA2
PTB16
PTD3
PTD2
D15
D14
AREF
GND
D13
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0