POLYTECH NICE SOPHIA

Robotique, Systèmes autonomes

UNIVERSITÉ CÔTE D'AZUR

Rapport final de projet

Année scolaire 2024-2025

NXPCup

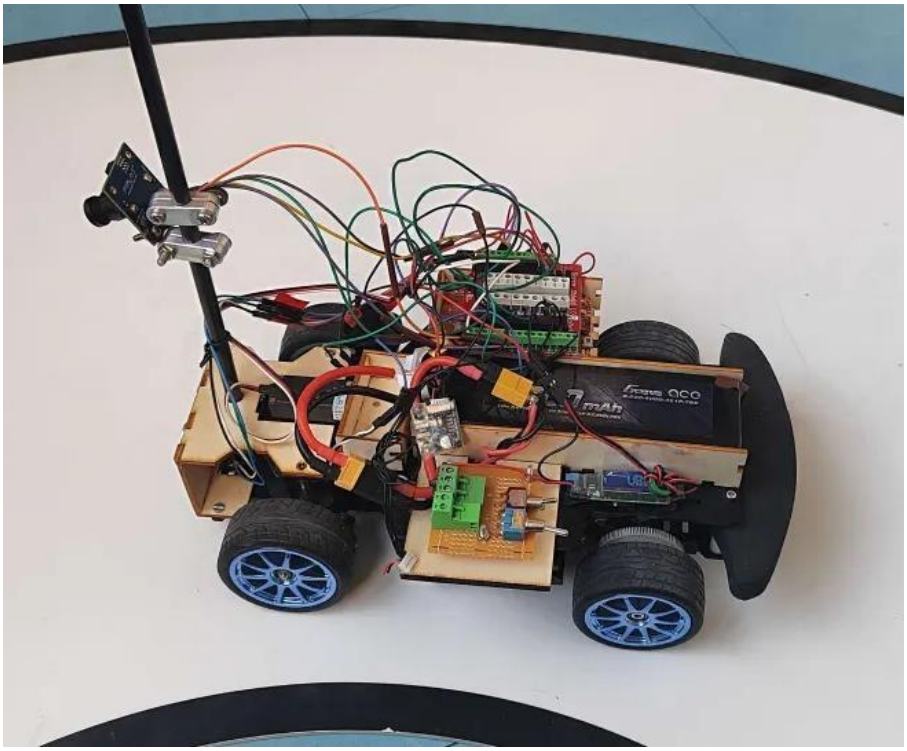


Figure 1 - Image du Robot

Etudiants : TIBAUDO Romain

Et

WARTSKI NARANJO Daniel

Ecole Polytechnique Universitaire de Nice Sophia-Antipolis, Département électronique 1645 route des Lucioles, Parc de Sophia Antipolis, 06410 BIOT

Sommaire

I. NXP Cup

La NXP Cup est une compétition technique destinée aux étudiants passionnés de robotique et d'ingénierie embarquée. L'objectif est de concevoir, assembler et programmer une voiture autonome capable de suivre un circuit avec rapidité, précision et fiabilité.

Chaque équipe peut être composée de 1 à 3 étudiants, provenant de n'importe quelle école, université, club robotique ou association STEM. Tous les niveaux d'études sont acceptés, du collège jusqu'au master, et les équipes peuvent mélanger des étudiants de différents établissements.

Côté matériel, le véhicule doit être propulsé par un maximum de quatre moteurs et équipé d'au plus quatre roues. La navigation doit obligatoirement se faire à l'aide d'un capteur optique (caméra), mais il est possible d'ajouter d'autres capteurs pour améliorer la détection de l'environnement. Tous les composants électroniques doivent être basés sur des microcontrôleurs ou processeurs NXP, qu'ils soient intégrés à des cartes existantes ou conçus sur mesure.

Il est impératif que le véhicule soit entièrement autonome pendant les phases de course. Aucun contrôle à distance ni connexion sans fil ne sont autorisés lors des épreuves officielles. Une seule batterie est autorisée pour alimenter l'ensemble du système, avec des limites strictes (par exemple, pour les batteries LiPo : 3 cellules, 11.1 V max, 6000 mAh). Des règles de sécurité spécifiques s'appliquent à leur transport et à leur utilisation lors des événements.

La course se déroule sur un circuit inconnu des participants jusqu'au jour J. Chaque équipe dispose de trois tentatives pour réaliser un tour complet, mais seul le premier tour réussi est pris en compte pour le classement. Après les deux premiers virages, un obstacle blanc (un cube de 20 cm) est placé sur la piste. La voiture doit ralentir et s'arrêter à moins de 10 cm de cet obstacle, sans le toucher, pour obtenir un bonus de 150 points.

Enfin, NXP fournit de nombreuses ressources pour aider les équipes, comme un Gitbook technique, un serveur Discord officiel, ainsi que l'accompagnement de référents sur les campus lors des événements.

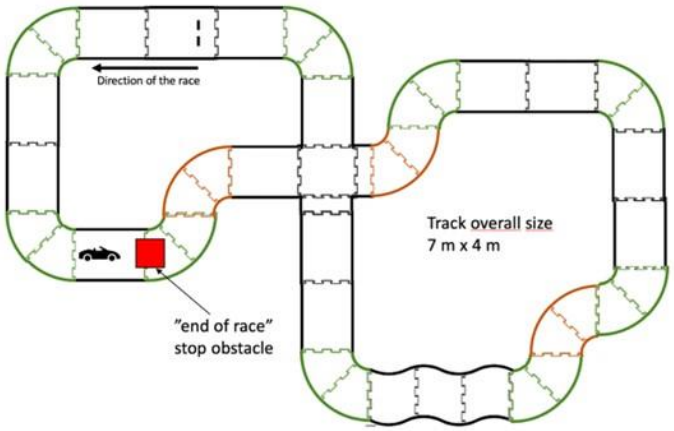


Figure 2 - Schéma Piste NXPCup

Concernant les règles de la course, la voiture doit garder au moins deux roues sur la piste tout au long de son trajet.

Après avoir atteint la ligne d'arrivée, la voiture doit visiblement ralentir, puis s'arrêter à 10 cm de l'obstacle.

II. Matériel

Liste du matériel :

Fonction	Composant	Description
Unité de contrôle	Carte NXP FRDM-K22F	Basée sur un processeur ARM Cortex-M4, compatible MCUXpresso, elle gère l'ensemble des capteurs, actionneurs et algorithmes.
Vision (suivi de ligne)	Caméra linéaire Parallax TSL1401-DB	Capture une ligne de 128 pixels ; permet une détection précise des lignes au sol pour le guidage.
Propulsion	2 moteurs brushless A2212/15T 930KV + ESC blheli_32	Fournissent la traction arrière. Chaque moteur est contrôlé par un ESC pour ajuster la vitesse via PWM.
Direction	Servomoteur Futaba S3010	Oriente les roues avant en fonction des corrections calculées par le contrôleur PID.
Détection d'obstacles	Capteur ultrason HC-SR04	Mesure la distance à un obstacle sur la piste, utilisé pour l'arrêt à 10 cm lors du second tour.
Alimentation — principale	Batterie LiPo 5000mAh, 11.1V (3C1P, 55.5 Wh)	Source d'énergie principale du système.
Alimentation — conversion	Power Module V6.0	Fournit du 7V pour les moteurs et du 5V pour la carte via régulation.
Alimentation — servomoteur	Régulateur Hobbywing UBEC 3A	Alimente spécifiquement le servomoteur avec un courant de 3A, indépendant de la carte.
Structure du robot	Châssis NXP avec kit de montage	Base mécanique du robot, conçue pour accueillir tous les composants.
Connectique	Câbles, connecteurs divers	Assurent l'interconnexion entre capteurs, actionneurs et carte de commande.

III. Circuit électrique

a. Connexion de la Line Scan Camera via ADC

La TSL1401-DB de Parallax est une caméra linéaire conçue pour l'acquisition d'images en une seule dimension, idéale pour des applications telles que le suivi de ligne. Elle fonctionne en transmettant les niveaux de luminosité détectés via une sortie analogique (A0), tandis que des broches numériques sont utilisées pour piloter les signaux de synchronisation nécessaires à la lecture des pixels. Elle est utilisée principalement pour le suivi de piste et la détection de la ligne d'arrivée pour ralentir le robot.

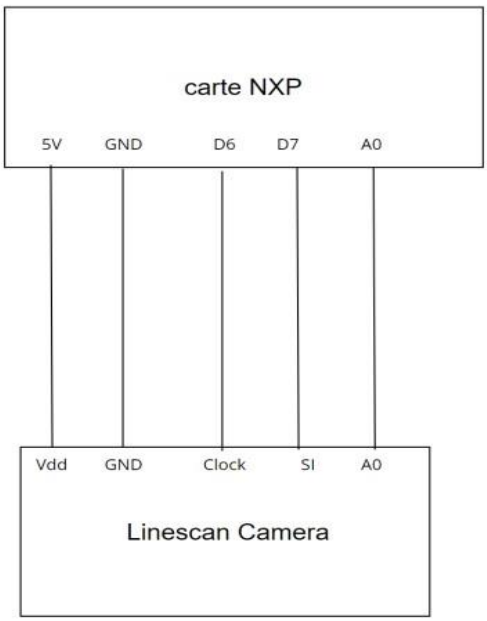


Figure 3 - Configuration pin Linescan

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 envoyées via l'ADC à chaque bord descendant de CLK
- Après la 18ème pulsation sur CLK, la phase d'intégration commence, sa longueur détermine l'exposition des pixels de la caméra
- Il est important de stopper les pulsations sur la CLK après la lecture des 128 pixels de la caméra

b. Connexion du capteur à ultrason hc-sr04

Le capteur à ultrasons permet de détecter la présence d'obstacles sur la piste. Son fonctionnement repose sur deux broches : la broche Trigger, qui envoie une impulsion pour initier la mesure, et la broche Echo, qui reçoit le signal réfléchi et permet de calculer la distance jusqu'à l'obstacle en mesurant le temps de retour de l'onde sonore. Il est utilisé notamment pour détecter l'obstacle de fin et arrêter le robot.

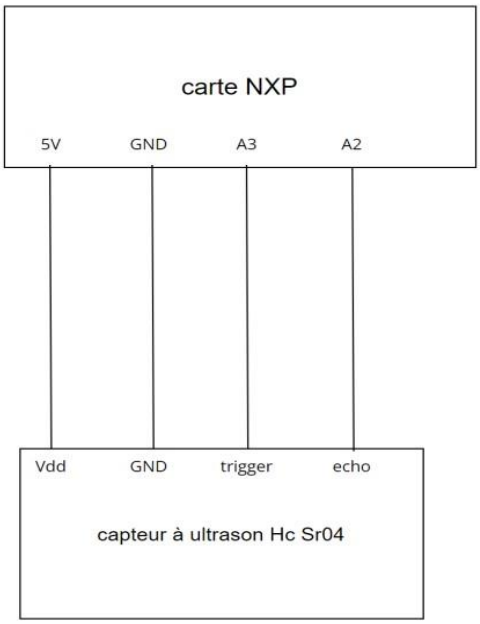


Figure 4 - Configuration pin HC-SR04

Cycle de fonctionnement :

Lorsque le pin Trigger est à l'état haut, le capteur envoie 8 impulsions à 40 Khz 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.

c. Connexion du servomoteur

Le servomoteur assure le contrôle de la direction du robot en orientant les roues avant. Il est piloté par un signal PWM à une fréquence de 50 Hz. Une seule broche est utilisée : la broche PWM, qui permet de définir l'angle de direction en ajustant le rapport cyclique du signal.

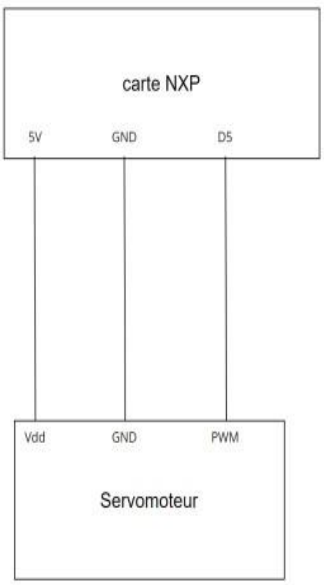


Figure 5 - Configuration pin servomoteur

d. Connexion des ESC

Les moteurs brushless assurent la propulsion du robot en entraînant les roues arrière. Leur vitesse est contrôlée par des signaux PWM transmis via des contrôleurs électroniques de vitesse (ESC). Chaque moteur reçoit un signal PWM à une fréquence de 48 kHz, généré par les modules FTM de la carte NXP. Les broches utilisées pour cette commande sont les suivantes :

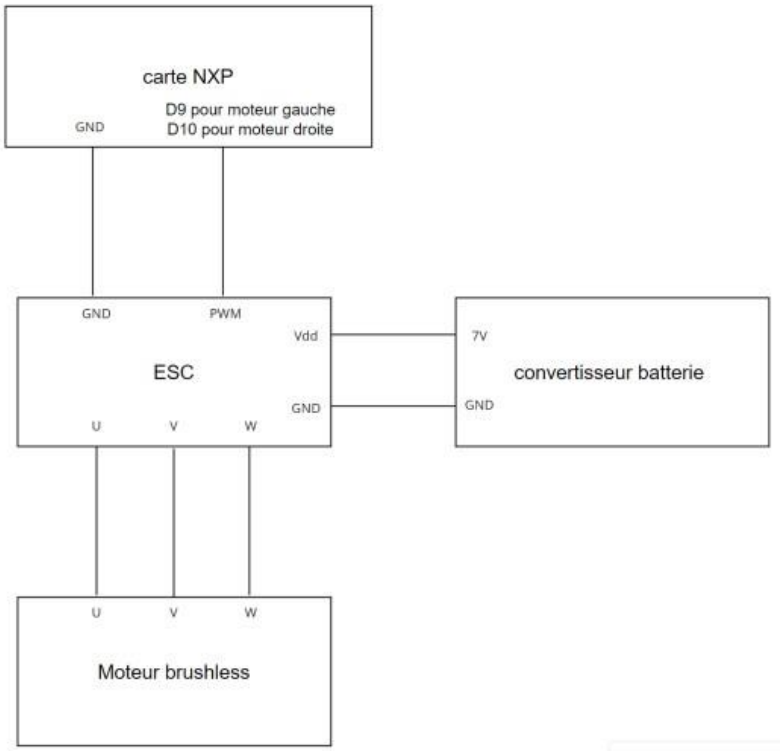


Figure 6 - Configuration pin moteur brushless

IV. Navigation

a. Fonctionnement du programme

1. Initialisation et configuration

Dans main(), tout le matériel est configuré et initialisé :

- ➔ PWM pour moteurs et servos via FTM0 (gauche/droite) et FTM2 (direction).
- ➔ ADC pour la caméra analogique (lecture des pixels).
- ➔ GPIO pour les capteurs (ultrason, bouton).
- ➔ SysTick pour les délais en microsecondes.

2. Perception — Suivi de ligne par caméra

Capteurs utilisés :

- Une caméra analogique linéaire est utilisée pour capturer 128 pixels via un signal d'horloge (clockPulse) et un signal SI.
- Un capteur ultrason est utilisé pour détecter la distance entre la voiture et un obstacle possible

Fonctionnement :

- getErrorFromCamera() :
  - Capture les 128 pixels via la fonction readADC()
  - Appelle detectTrack() pour détecter deux lignes noires (bordures de la piste).
  - Calcule le centre de la piste.
  - Retourne l'erreur de position : la différence entre le centre de l'image et celui de la piste.
- detectTrack() :
  - Scanne les pixels pour trouver deux séquences de pixels noirs avec une bande blanche entre elles.
  - La position des lignes noires gauche et droite permet de déduire la trajectoire à suivre.

3. Contrôle — Ajustement de direction (PID)

Le programme utilise un contrôleur PID dans la fonction turn() pour ajuster la direction. Cette fonction se structure de la façon suivante :

- Utilise un contrôleur PID pour calculer l'angle de correction basé sur l'erreur actuelle, l'intégrale des erreurs passées et la dérivée de l'erreur.
- Un mécanisme anti Wind-up est mis en place par une limite appliquée à l'intégrale.
- Ce calcul est ensuite utilisé pour ajuster le rapport cyclique (PWM) du servomoteur (direction).
- Les variables turnLeft, turnRight, middle servent à stocker l'état de la manœuvre.
- Les valeurs de updatedDutycycleTurn, updatedDutycycleLeft et updatedDutycycleRight sont mises à jour selon l'état des variable précédentes.
- Le robot ralentit un peu la roue interne dans un virage, par exemple, lorsque turnLeft est true, le updatedDutycycleLeft sera plus faible que updatedDutycycleRight.
- En ligne droite, il accélère, lorsque middle est true, les 2 variables sont égales.

4. Commande moteur (vitesse)

- UpdatePwmDutycycleSpeed() applique les valeurs PWM aux deux moteurs brushless selon la valeur de updatedDutycycleLeft et updatedDutycycleRight.
- UpdatePwmDutycycleTurn() applique les valeurs PWM au servomoteur selon la valeur de updatedDutycycleTurn.

5. Évitement d'obstacles — Capteur ultrason

Basé sur un capteur HC-SR04, contrôlé par :

- sendTrigger() : envoie une impulsion.
- ECHO_IRQ_HANDLER() : mesure le temps de réponse et calcule la distance en centimètres à partir du temps.

Fonctionnement :
Si distance < 20 cm : motorBreak() est appelé, le robot s'arrête puis attend que l'obstacle soit dégagé ou qu'un bouton soit pressé pour redémarrer.

6. Boucle principale

Dans main(), la boucle while (1) contient :

- La gestion périodique du capteur ultrason.
- La lecture de la caméra et le calcul d'erreur.
- L'ajustement dynamique de la direction avec turn().
- L'adaptation de la vitesse en fonction de la trajectoire.

b. Contrôles moteurs

Moteurs 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 plage de fonctionnement de nos moteurs (le dutycycle du PWM). De manière expérimentale nous avons déterminé que la plage 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 nos moteurs sont arrêtés pour 47.

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 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.

V. Algorithme

a. Améliorations apportées

Notre groupe a pu apporter plusieurs améliorations au projet :

- Amélioration de la logique de la fonction detectTrack() qui auparavant détectait une intersection sur les premiers pixels de la caméra si le sol était noir.
- Amélioration de la logique de la fonction getErrorFromCamera() qui pour compenser l'erreur précédente tournait à droite uniquement dans le cas où la bordure gauche n'était pas détectée.
- Amélioration de la logique de la fonction turn() qui utilisait seulement 3 positions pour le servo moteur. Quand le robot devait tourner les roues prenaient un angle de 45° et quand le robot devait aller tout droit elles prenaient un angle de 0°. On a donc remplacé cette logique par un contrôleur PID qui corrige l'erreur obtenue avec getErrorFromCamera().

1. Changements apportés à detectTrack()

La fonction detectTrack() a subi de gros changements, en effet elle fonctionnait de cette façon :

- ➔ Les bordures avaient par défaut une valeur de -1
- ➔ La fonction parcourait chaque pixel du tableau
- ➔ Comptait les pixels consécutifs sous le seuil THRESHOLD_BLACK
- ➔ Quand 8 pixels noirs consécutifs étaient détectés, la bordure gauche était positionnée à la position du 8-ème pixel, quand 8 autres pixels noirs étaient détectés, avec une distance minimum entre le premier et la bordure gauche, la bordure droite était placée à la position du premier pixel.
- ➔ Si une seule bordure était trouvée, sa position déterminait s'il s'agissait du bord droit ou gauche.

Cette fonction créait plusieurs problèmes, le premier étant que si le sol autour de la piste est sombre, le robot placera toujours la bordure gauche à la position 0 et la seconde étant que si seule la bordure gauche était trouvée et que celle-ci était positionnée à droite de la caméra, le robot tournait dans le mauvais sens car il pensait que c'était la bordure droite.

La nouvelle fonction règle ces problèmes, en effet maintenant, le programme attend de trouver la piste blanche pour déterminer la position de la bordure gauche, de plus, si une seule bordure est détectée, le bord de la caméra qui n'est pas blanc indiquera de quel côté de la piste se trouve la bordure.

2. Changements apportés à getErrorFromCamera()

La fonction getErrorFromCamera() a subi de gros changements, en effet elle fonctionnait de cette façon :

- ➔ Une première partie qui sert à lire les valeurs de la linescan caméra

➔ Une deuxième partie qui calcule l'erreur et appelle adjustDirection() pour passer la commande au servomoteur

C'est cette deuxième partie qui nous intéresse car due aux problèmes de l'ancienne fonction detectTrack(), celle-ci avait aussi de gros défauts, le principal étant que dans le cas où la bordure gauche était positionnée à 0, l'erreur se voyait attribuer la valeur arbitraire de 20. Cela compensait les problèmes de l'ancienne fonction detectTrack() en permettant au robot de tourner à gauche sans vraiment détecter la position de la bordure gauche.

Nous avons changé ça dans la nouvelle version en supprimant les comportements arbitraires dans les cas spéciaux.

3. Changements apportés à Turn()

La fonction Turn() avait une logique très basique, en effet celle-ci se contentait de donner à updatedDutycycleTurn la valeur 8 si l'erreur est négative et inférieure à -2, la valeur 13 si elle est positive et supérieur à 2 et la valeur 11 dans les autres cas.

Cette logique donnait un contrôle très binaire sur la direction du robot. On l'a donc remplacé par un correcteur PID avec anti wind-up sur l'erreur, qui associé à un mapping linéaire de la consigne du servo-moteur permet un contrôle de la direction bien plus stable et précis.

b. A améliorer

Il reste cependant des choses à améliorer :

- Sensibilité trop élevée à la lumière, la caméra linescan est par défaut très sensible aux conditions d'éclairage. De plus le programme utilise pour l'instant des seuils fixes pour la détection du noir et du blanc ce qui fait que si la pièce n'est pas très bien éclairée, le robot ne détectera pas les bords de la piste.

- Amélioration des constantes du PID

- Utilisation d'un seul capteur, le robot n'utilise que la linescan caméra pour suivre la piste, cela le rend extrêmement dépendant. Si la calibration ou les conditions d'éclairage ne sont pas suffisantes pour la linescan caméra le robot ne pourra pas opérer.

- Détection de la ligne de fin, pour l'instant la linecam s'occupe de la détection de la ligne de fin. Cependant, cela peut interférer avec le programme de navigation et, dans certains cas, le robot tournera au lieu de simplement ralentir.

VI. Annexe

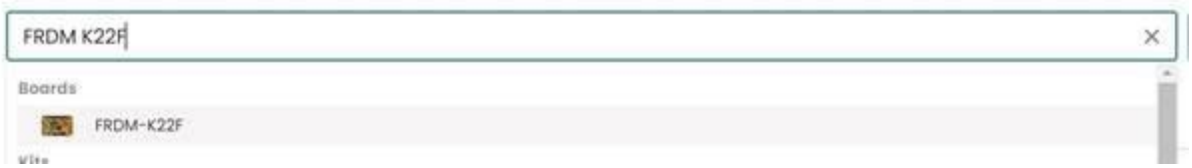
Liens utiles :

- 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/wphttps://futabausa.com/wp-content/uploads/2018/09/S3010.pdfcontent/uploads/2018/09/S3010.pdf

Téléchargement :

- MCUxpresso https://www.nxp.com/design/design-center/software/developmenthttps://www.nxp.com/design/design-center/software/development-software/mcuxpresso-software-and-tools-/mcuxpresso-integrated-development-environment-ide:MCUXpresso-IDEsoftware/mcuxpres so-software-and-tools-/mcuxpresso-integrated-developmenthttps://www.nxp.com/design/design-center/software/development-software/mcuxpresso-software-and-tools-/mcuxpresso-integrated-development-environment-ide:MCUXpresso-IDEenvironment-ide:MCUXp resso-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

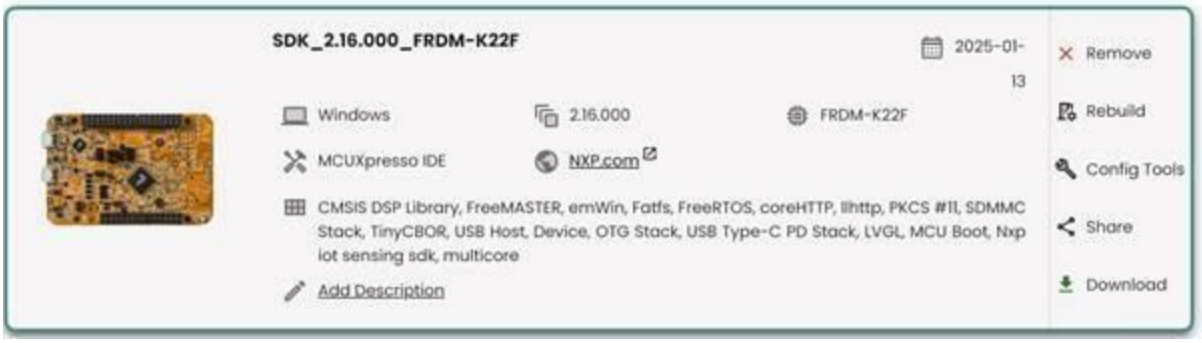
    - Se connecter à NXP (créer un compte si vous n'en possédez pas)
    - Dans la barre de recherche entré FRDM K22F



    - Cliquer sur la carte dans l'onglet Board

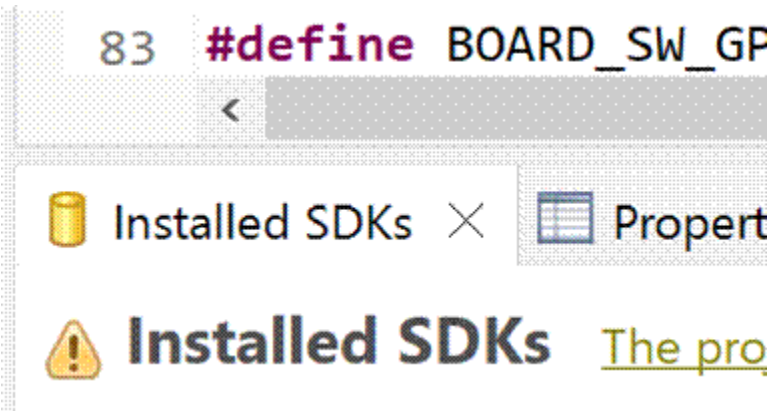
Cette page devrait se trouver en dessous (il faut scroller vers le bas)

- 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 :