

# Choix des composants pour les projets Mécatro 2023 – updaté 2024

Matthieu Vigne

Ce document propose une liste de composants pour les projets Mécatro: un robot suiveur de ligne et un pendule inverse style Segway. Le choix des composants est principalement guidé par la contrainte suivante: la recherche d'un système plug-and-play ne nécessitant aucune soudure ni branchements par connecteurs Dupont non polarisés (pour limiter les erreurs, débranchements intempestifs...).

## Microcontrôleur

L'élément central du robot est bien sur son processeur. Deux choix classiques s'offrent à nous: Raspberry Pi ou Arduino. Je privilégie le choix d'une Arduino, bien plus simple à prendre en main: pas besoin de s'embêter avec une image linux, une installation de dépendance, un environnement de développement et une chaîne de compilation... Quoi que bien plus limitée, les performances d'une Arduino sont bien suffisantes pour ce projet. De plus, l'écosystème Arduino en terme de robotique est plus développé que celui de la Raspberry Pi. Une Raspberry Pi pourra toujours être rajoutée par la suite, selon un schéma maître-esclave classique: la Raspberry effectue les calculs haut-niveau complexes (telle que le traitement de données de caméra ou LIDAR) et envoie des ordres à une Arduino responsable de contrôler en bas-niveau les actionneurs et capteurs du robot.

Le seul inconvénient que je vois à l'utilisation d'une Arduino est un debug plus compliqué: pour une Raspberry, équipé d'un système de fichier, réaliser une télémétrie (enregistrement de données pour analyse ultérieure) est très simple et ne pose pas de limite pratique. Une Arduino à l'inverse ne dispose pas de mémoire : les données doivent donc être streamées vers un PC hôte, avec une limitation de bande passante et de temps de traitement plus significative. L'ajout d'un module WiFi pourra être envisagée pour le cas du pendule inverse, où l'utilisation d'un câble pour le debug serait peu pratique.

Concrètement, c'est un clone d'Arduino que je propose d'utiliser, afin de répondre à la problématique de connexion simple des capteurs, ne nécessitant pas un cablage sur les GPIOs. Pour cela, l'idée est d'utiliser des capteurs communiquant par bus I2C: un protocole de communication série très simple, quoi que peu performant, ne nécessitant que deux fils (données et horloge). Sur un bus I2C, un unique maître (l'Arduino) communique avec N esclaves, chacun disposant d'une adresse unique pour l'identifier: ainsi, de nombreux capteurs peuvent être branchées en série, et facilement rajoutés au cours de l'évolution du projet.

Afin de faciliter le branchement des composants I2C, plusieurs constructeurs de périphériques Arduino ont proposé un système de connection plug-and-play. Ici, je propose d'utiliser comme carte microcontrôleur la [Sparkfun Redboard QWIIC](#), un clone d'Arduino Uno directement équipé d'un port *QWIIC*: un port I2C utilisant un connecteur JST. Ceci permet de brancher facilement les cartes *QWIIC* fabriquées par Sparkfun.

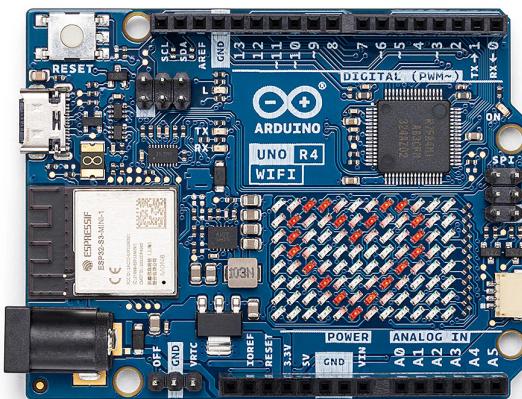


Figure 1: [Arduino Uno R4 WiFi](#)

Par ailleurs, le même système a été développé par Adafruit, qui le nomme *Stemma QT*: il s'agit du même connecteur, les deux sont directement compatibles. Enfin, d'autres cartes peuvent être trouvées chez Seedstudio, sous le nom de *Grove* - à noter que le connecteur est différent et nécessite un [cable d'interface](#)

**Update 2024 : on passe cette année à une [Arduino Uno R4 WiFi](#), pour pouvoir faire de la télémétrie sans câble.** Elle possède toujours le même connecteur QWIIC, mais désormais deux micro-processeurs : un RA4M1 chargé de faire les calculs, et un ESP32-S3 pour la communication WiFi. Les deux processeurs communiquent via un port série avec un protocole UART, dont la fréquence est a priori fixée par Arduino. Afin de changer cette fréquence, nous avons écrit un nouveau firmware, qui doit être flashé sur chaque carte avant utilisation (normalement toutes les cartes ont été flashées avant le début du projet).

Une autre différence est à mentionner: les niveaux de tension utilisées. Arduino fonctionne classiquement sur un niveau 5V, mais de nombreux composants modernes, ainsi que les processeurs style Raspberry Pi, fonctionnent de 3.3V. Un grand nombre de capteurs sont compatibles avec les deux niveaux de tension, mais ce n'est pas le cas de tous, et c'est un point qu'il convient de vérifier: un composant 5V opéré en 3.3V ne fonctionnera pas, tandis qu'un composant 3.3V, s'il est branché en 5V, va tout simplement griller ! En pratique, la carte Sparkfun Redboard peut opérer en 3.3V ou 5V au niveau de ses GPIO - par contre, l'interface I2C QWIIC est **toujours** en 3.3V. A l'inverse, le système Grove était à l'origine prévu pour des composants 5V, même si bon nombre de composants sont compatibles avec les deux niveaux. C'est un point à vérifier pour chaque composant ; en cas de besoin, de convertisseurs de niveau logique [comme celui-là](#) existent bien entendu.

## Motorisation

Une fois le processeur choisi, l'élément le plus structurant dans la conception d'un robot est le choix de sa motorisation: celui-ci va en effet dimensionner la conception mécanique (moteur, taille des roues...) et électrique (contrôleur de moteur, codeur, batterie).

Trois grandes familles de moteurs existent dans le domaine de la robotique de loisir:

- les moteurs à courant continu à balais: ce sont de loin les plus simples, les moins chers et les moins performants. L'absence de besoin de commutation ou de régulation de courant rend leur contrôle basique très simple (un simple relai peut les faire tourner) ; leurs performances sont cependant médiocres, en partie due à la faible qualité des réducteurs bon-marchés utilisés en sortie (un moteur classique tournant à plusieurs milliers de tour minute nécessite un réducteur de rapport de réduction élevé).
- les moteurs pas à pas: principalement utilisé dans des applications à faible charge, ils permettent un positionnement précis mais délivrent peu de couple. Ces moteurs sont utilisés partout en impression 3D, car ils permettent un positionnement simple sans nécessiter de capteurs. Appliqués à un robot mobile cependant, leur faible couple limitera l'accélération du système. De plus, étant par nature contrôlés en position, ils ne nécessitent pas d'asservissement, ce qui rend obsolète toute une partie du projet.
- les moteurs à courant continu sans balais: évolution significative des versions sans balais, ils offrent des performances bien supérieures (avec des rapports poids-puissance plus de 10x supérieurs, quand on prend en compte le réducteur nécessaire aux moteurs à balais). Ils sont aussi beaucoup plus cher (tant le moteur que l'électronique de puissance), et encore peu disponibles en tant que composants standards pour la robotique de loisir (même si cette technologie se démocratise ces dernières années). Leur contrôle est logiquement bien plus complexe (même si un contrôleur de moteur intégré peut l'effectuer en mode boîte noire).

Pour commencer par quelque chose de simple, les projets seront à base de moteur à courant continu. Il convient donc de choisir un moteur, contrôleur de moteur et codeur adapté. Ce choix se révèle assez ardu, et finalement très empirique, à cause d'un simple phénomène: les frottements dans la transmission. Les moteurs à balais nécessitent un fort rapport de réduction, l'ensemble moteur + réducteur, appelé motoréducteur, est souvent vendu d'un seul bloc. Or, pour des petits moteurs, peu onéreux, le rendement des réducteurs est souvent très faible, ce qui peut poser des problèmes à faible couple / faible vitesse. Ainsi, par le passé, je me suis souvent trouvé obligé de sur-dimensionner les moteurs, et d'essayer plusieurs modèles par essai-erreur

jusqu'à en trouver un aux performances satisfaisantes.

Pour palier cette limitation, je propose de séparer au maximum l'inter-dépendance entre composants. Une possibilité serait en effet de choisir un motoréducteur équipé d'un encodeur intégré, puis de prendre un contrôleur de moteur tout juste adapté en terme de puissance. Mais alors, si le moteur s'avère insuffisant, c'est l'entièreté du système qu'il faut revoir, avec potentiellement des difficultés d'intégration du nouveau codeur / contrôleur. A l'inverse, je propose d'utiliser un unique codeur externe, et un contrôleur de moteur de puissance suffisante pour s'adapter à n'importe quel moteur de taille raisonnable. Ainsi, le motoréducteur est isolé, et pourra facilement être changé par la suite (tant dans la phase de design initiale que comme potentiel d'amélioration, si les élèves veulent équiper leur robot de moteurs plus puissants / rapides. A noter qu'un vrai gap technologique serait franchis avec l'utilisation de moteur sans balais, mais cela demande une étude plus approfondie).

## Moteur et roue

Le choix du moteur dépend évidemment de la roue utilisée, celle-ci agissant comme un rapport de réduction. Je propose d'utiliser les [roues en plastique de Pololu](#): il s'agit de toute une gamme de roue en plastique, avec un pneu en caoutchouc, avec un très bon rapport qualité-prix: l'adhérence est décente, et les roues tournent autour de 5€ l'unité. Ces roues existent en plusieurs diamètre (de 37 à 100mm), et peuvent se monter facilement sur différents diamètres d'arbres grâce à des [moyeux en aluminium](#): ainsi, il est facile de changer de taille de roue si nécessaire.

Mon expérience personnelle me pousse à privilégier un motoréducteur avec un rapport de réduction plus faible, afin de limiter les pertes de rendement dans le réducteur. Ainsi, une roue de faible diamètre est à privilégier ; une limite apparaît cependant lié à la taille du moteur: je vise des moteurs faisant un diamètre de l'ordre de 40mm, aussi, une roue de 70mm semble adaptée (permettant une hauteur de chassis de l'ordre de 15mm).

Il s'agit maintenant de sélectionner un moteur pour nos deux applications. Dans le cas de la voiture, l'estimation de la vitesse de rotation nécessaire est assez simple: il suffit de se fixer une vitesse consigne à atteindre. 1m/s me semble déjà un point de départ ambitieux: pour un robot d'environ 30cm de long, cela correspond à trois fois la longueur de son corps par seconde. Cette vitesse linéaire correspond à 275rpm. Pour le couple, en considérant un véhicule de 2kg accélérant à  $1m/s^2$ , une force de poussée de 2N est nécessaire, soit 0.035Nm par moteur, c'est à dire 0.35kg.cm. Il s'agit bien sur uniquement d'un ordre de grandeur, qui suppose un actionnement parfait sans frottement.

Le dimensionnement pour le projet du pendule inverse est plus difficile à réaliser a priori. En effet, en considérant le cas d'école d'un système sans frottement, on se rend compte qu'il ne faut que très peu de couple, et très peu de vitesse, pour stabiliser le système au voisinage de son point d'équilibre. La présence de frottements, jeu, retard... vont complexifier les choses. Finalement, je propose de considérer un ordre de grandeur du couple nécessaire correspondant à un angle maximal de ratrapage: il faut que les moteurs puissent vaincre le couple créé par la gravité à cet angle. Fixons arbitrairement un seuil à 10 degré, et considérons à nouveau un véhicule de 2kg. Pour connaître le couple de gravité, il faut connaître la position du centre de masse du système. Noter qu'il s'agit d'une variable d'ajustement intéressante pour le design: rapprocher le centre de masse de roues, en diminuant l'inertie du système et le couple induit par la gravité, diminue le couple moteur nécessaire, mais accélère la chute du robot (en réduisant la période du pendule associé). En pratique je pense fixer le centre de masse assez près des roues (l'effet bénéfique en couple me semble supérieur aux inconvénients liés à la dynamique, qui peuvent être mitigés en augmentant la fréquence de contrôle). Mais mieux vaut être pessimiste pour le dimensionnement: je propose de considérer un centre de masse à 10cm de l'axe des roues. Le couple créé par la gravité à 10 degré d'inclinaison est alors de 0.34Nm, soit 1.7kg.cm par moteur. A noter que le diamètre des roues n'entre pas en compte ici.

De nombreux fabricants de petits motoréducteurs existent: ceux que je connais le plus sont Pololu, ServoCity,



Figure 2: Roue Pololu

MFA ; bon nombre sont distribués sur Gotronic ou Robotshop, qui sont des bons endroits où commencer à chercher.

De façon assez arbitraire, je propose de commencer par la gamme de moteur [37mm de Pololu](#): il s'agit du même moteur monté sur un réducteur à engrenages droits. Très peu cher, mon expérience reste celle d'un rendement assez moyen - moins bon que ce que j'ai pu trouver chez des moteurs avec train épicycloidal de Servocity par exemple - mais, je pense, suffisant pour réaliser ces projets. Plus précisément, je propose de prendre le [4743](#) avec un rapport de réduction de 50 pour le pendule inverse:, 100 rpm, 10kg.cm @ max power

## Controleur de moteur

Le point suivant consiste à trouver des contrôleurs de moteur capables de s'adapter à différents moteurs. Concrètement, la variable principale est le courant maximum disponible: pour cette taille de moteur, on s'intéresse comme ordre de grandeur avec un courant continu moyen de l'ordre d'1A, avec d'éventuelle pointes autour de 5A.

Il existe des cartes intégrant un microcontrôleur avec un pont en H : ainsi, l'asservissement peut être fait en déporté sur cette carte. L'intérêt pédagogique d'un tel système me semble cependant beaucoup plus faible: finalement, le problème de contrôle est déjà résolu, et la seule difficulté éventuelle réside dans la communication avec cet esclave. Aussi, c'est un simple pont de H que je cherche.

Mon choix se porte sur le [Cytron 10A 7-30V Dual Channel DC Motor Driver Shield](#) pour plusieurs raisons:

- il fournit une puissance plus que suffisante (10A continu, 30A pic!) pour un prix très faible: 30€ pour deux moteurs. Ceci est tout simplement du à une conception moderne avec des transistors MOS performants: à titre de comparaison, des contrôleurs utilisant le vénérable L298, sorti en 2000, ne propose que 2A continue, 4A pic - pourtant, il est vendu (monté sur circuit) entre 5€ et 30€ ! A noter que le fait que le contrôleur puisse monter bien plus haut en puissance n'est pas un problème: en pratique, c'est la résistance du moteur qui limitera le courant ; le contrôle se fait par modulation du rapport cyclique (duty cycle) du pont en H, et non par injection de courant.
- le format shield est très pratique: pas besoin de câbles ; il fournit de plus une alimentation directe à l'Arduino. Autre fonctionnalité sympathique, des boutons sur la carte permettent de tester le branchement du moteur "à la main" sans avoir besoin de code.
- le contrôleur est un simple pont en H, sans intermédiaire (contrairement au [contrôleur de moteur QWIIC](#)). Ainsi, l'utilisateur pilote directement, depuis l'Arduino, le signal PWM du moteur. Ceci permet un contrôle du courant envoyé *sans aucune latence* - plus précisément, le temps de réponse du signal de contrôle est directement celui de la fréquence de modulation (10kHz-20kHz), prévu pour être bien supérieur à celui du moteur. Pédagogiquement, je trouve ça bien plus intéressant de pouvoir détailler ce qui se passe, au lieu d'une "boîte noire"

## Codeur

Le dernier point à traiter pour la motorisation est le capteur mesurant l'angle du moteur, le codeur rotatif. Une première solution consiste à monter un codeur incrémental (hall ou optique) sur un axe sortant à l'arrière du moteur. Bien que très pratique niveau intégration, cette solution présente trois inconvénients majeurs:

- le codeur incrémental est à brancher sur 4 broches de l'Arduino (alim + signaux A/B) avec des connecteurs Dupont, alors que notre but est d'éviter tout problème de cablage.
- le fait de devoir compter les pas prend de la puissance de calcul à l'Arduino - ce qui peut être non négligeable pour des vitesses de rotation élevées (à 12bit 300rpm, on parle de 20000 interruptions /

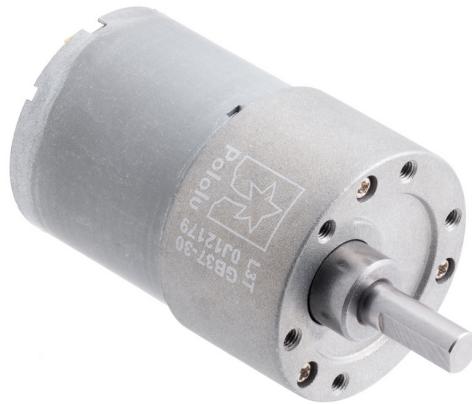


Figure 3: Moteur Pololu 37mm

seconde / codeur)

- le codeur étant intégré au moteur, changer de moteur oblige à changer de codeur ; de plus, tous les moteurs ne sont pas équipés d'axe arrière, réduisant le choix disponible

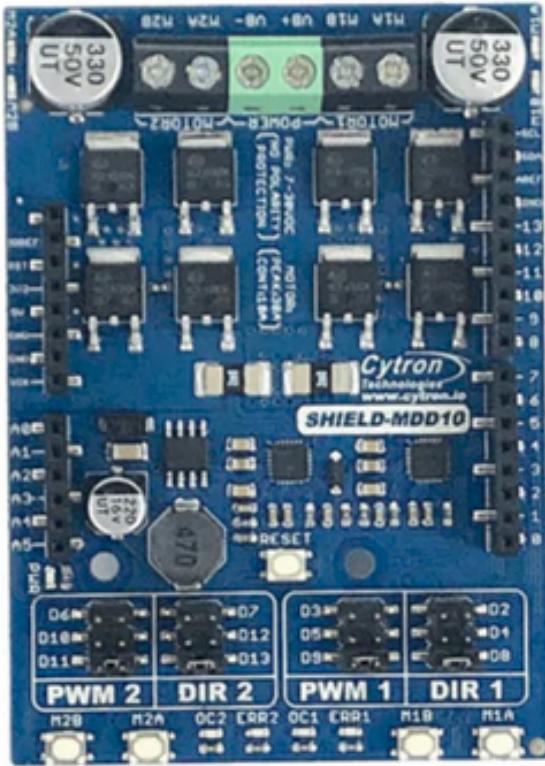


Figure 4: [Cytron 10A 7-30V Dual Channel DC Motor Driver Shield](#)

Pour ces raisons, je privilégie le choix d'un codeur externe, à monter sur l'arbre directement, et disposant d'une interface I2C, pour être utilisé simplement au sein du système QWIIC. La solution retenue - la seule trouvée en réalité - est [le codeur magnétique AS5600](#). Il s'agit d'un codeur absolu simple tour 12bit communiquant en I2C. Bien que sa forte compacité et son faible prix soient des atouts majeurs, il vient avec deux petits inconvénients :

- il utilise un connecteur Grove et non QWIIC, nécessitant un [cable d'interface](#).
- l'adresse I2C n'est pas réglable, ce qui signifie que deux codeurs ne peuvent pas opérer sur le même réseau. L'utilisation d'un [multiplexeur I2C](#) est donc nécessaire.

Ce codeur nécessite un aimant polarisé radialement, fixé sur l'axe moteur. La fixation peut se faire à l'aide du même moyeu que pour la roue. Ces aimants sont très peu onéreux, quoi que difficile à trouver par un revendeur européen. Je propose le site Néerlandais [magnetenspecialist](#) pour des aimants de 6x2.5mm à environ 0.50€ l'aimant.

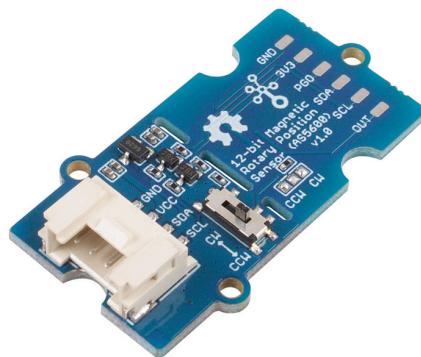


Figure 5: [Codeur magnétique AS5600](#)

## Capteurs

Enfin, il reste à choisir les capteurs spécifiques aux tâches de chaque robot - les codeurs, commun aux deux robots, ont déjà été évoqués.

### Pendule inverse: IMU

Le pendule inverse doit disposer d'une IMU pour sa stabilisation. Plusieurs versions sont proposées par Sparkfun - le travail à 3.3V permettant un large choix de composant. Une seule de ces options cependant est équipée d'un magnétomètre 3 axes - en plus du gyro et accéléromètre 3 axes. Il s'agit de l'[ICM-20948](#). Bien que ce capteur ne soit pas absolument nécessaire, il peut s'agir d'un plus appréciable - typiquement pour expérimenter de l'estimation et du contrôle de lacet. L'IMU ne présente par ailleurs pas d'inconvénient majeurs: elle présente un niveau de défauts et de bruit un peu supérieur à la gamme LSM6D d'ST, une autre alternative, mais cette dernière n'a pas de gyroscope - et je ne pense pas que la différence soit très significative.

**Update 2024: on passe à la [BMI-270](#) de chez Bosch, toujours sur une board Sparkfun, suite à de nombreux problèmes en 2023 : biais d'accéléro et de gyro parfois gigantesques, capteurs qui renvoient une constante jusqu'au reboot... Plus de magnéto, de toutes façons inutilisé l'année dernière, mais, on l'espère, des mesures de meilleure qualité.**

### Voiture: suivi de ligne

Pour la voiture, il s'agit d'opérer un suivi de ligne, à l'aide de phototransistors et de diodes pour l'éclairer. Bien qu'en théorie un seul capteur soit nécessaire, le fait de disposer de plusieurs capteurs en ligne permet une information plus riche - et offre plus d'options de contrôle de trajectoire.

Malheureusement, je n'ai pas trouvé, pour l'instant, de suiveur de ligne plug-and-play. Le meilleur compromis que j'ai est le [Line Follower Array](#) de Sparkfun: une rangée de 8 phototransistors (avec des leds), qui communique en I2C. La carte ne dispose malheureusement pas de port QWIIC, et nécessitera donc [un adaptateur à souder](#) (opération qui peut être faite avant le début du projet, ou par les élèves). A noter que la carte est affichée pour du 5V, mais le schéma fait référence à une opération 3.3V. J'ai envoyé un mail à Sparkfun pour une confirmation de compatibilité, et voir s'ils ont une alternative à proposer.

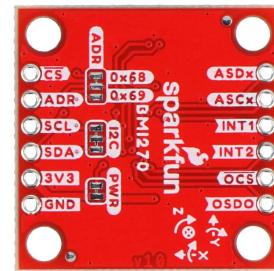


Figure 6: [IMU BMI-270](#)

### Inégration: latence et vitesse de la trame I2C.

Chaque composant individuel du robot a été identifié, il convient maintenant de vérifier que l'ensemble va fonctionner. Ici, le point limitant présent est le bus I2C: l'I2C est en effet un protocole assez lent, brancher trop d'esclaves dessus va donc nous forcer à baisser la fréquence de rafraîchissement des capteurs - et donc la fréquence de contrôle.

Nous allons donc estimer ici le temps de communication nécessaire. Le robot le plus gourmand sera le pendule inverse, l'IMU nécessitant plus de bande passante que le suiveur de ligne.

La vitesse maximale d'I2C supportée par une Arduino est une horloge à 400kHz. Nous nous intéressons ici uniquement au mode lecture - l'écriture de registre de configuration n'étant fait qu'au démarrage n'est pas critique. Pour communiquer avec un esclave, il faut envoyer son adresse d'écriture, l'adresse de registre à lire, et l'adresse de lecture - donc 3 octets. A cela s'ajoute start, restart et stop, ainsi que les acknowledgements à chaque octet, que pour simplifier nous comptons tous comme 1bit (en pratique acknowledge peut être plus long). Ainsi, un total de 30bits d'overhead est présent à chaque changement d'esclave. Pour l'IMU, les registres des trois capteurs sont consécutifs, et peuvent être lu d'une seule traite. Chaque capteur comprend 3 axes sur 2 octets chacun, soit 18 octets. Il faut compter 9 bits par octet ici, à cause du bit acknowledge. Il y a donc un total de  $30 + 18 * (8 + 1) = 156$  bits. Avec une horloge de 400kHz, cela correspond à 390us de temps de lecture.

Pour lire les codeurs, il faut lire 2 octets, donc à nouveau  $30 + 2 * (8 + 1) = 48$  bits = 120us. De plus, il faut compter le temps que va prendre le multiplexeur I2C à changer de bus (les deux codeurs ayant la même adresse): il s'agit d'une écriture de registre, pour 30 bits à nouveau, soit 75us.

Au total, on obtient un temps de communication de 700us. Il s'agit ici d'un calcul assez optimiste: en pratique, l'Arduino ne pourra pas traiter instantanément les trames pour en envoyer des nouvelles. Le temps effectif de lecture sera donc plus long ; ainsi, un contrôle à 1kHz (1ms de boucle) ne semble pas possible avec cette approche<sup>1</sup>. Mais atteindre 500Hz me semble possible - et mon expérience personnelle, sur ce genre de projet notamment, me laisse penser que cette période de contrôle sera suffisante (même si, il est vrai, augmenter la fréquence de contrôle améliore les performances).

## Bilan

La Figure 7 résume le fonctionnement général et l'inter-connection des éléments du robot (**schéma non à jour 2024**).

---

<sup>1</sup>On peut l'envisager en considérant des mises à jour partielles de capteurs, e.g. le magnétomètre qui n'est mis à jour qu'à 100Hz, ou encore lire les codeurs une itération sur deux en alternance.

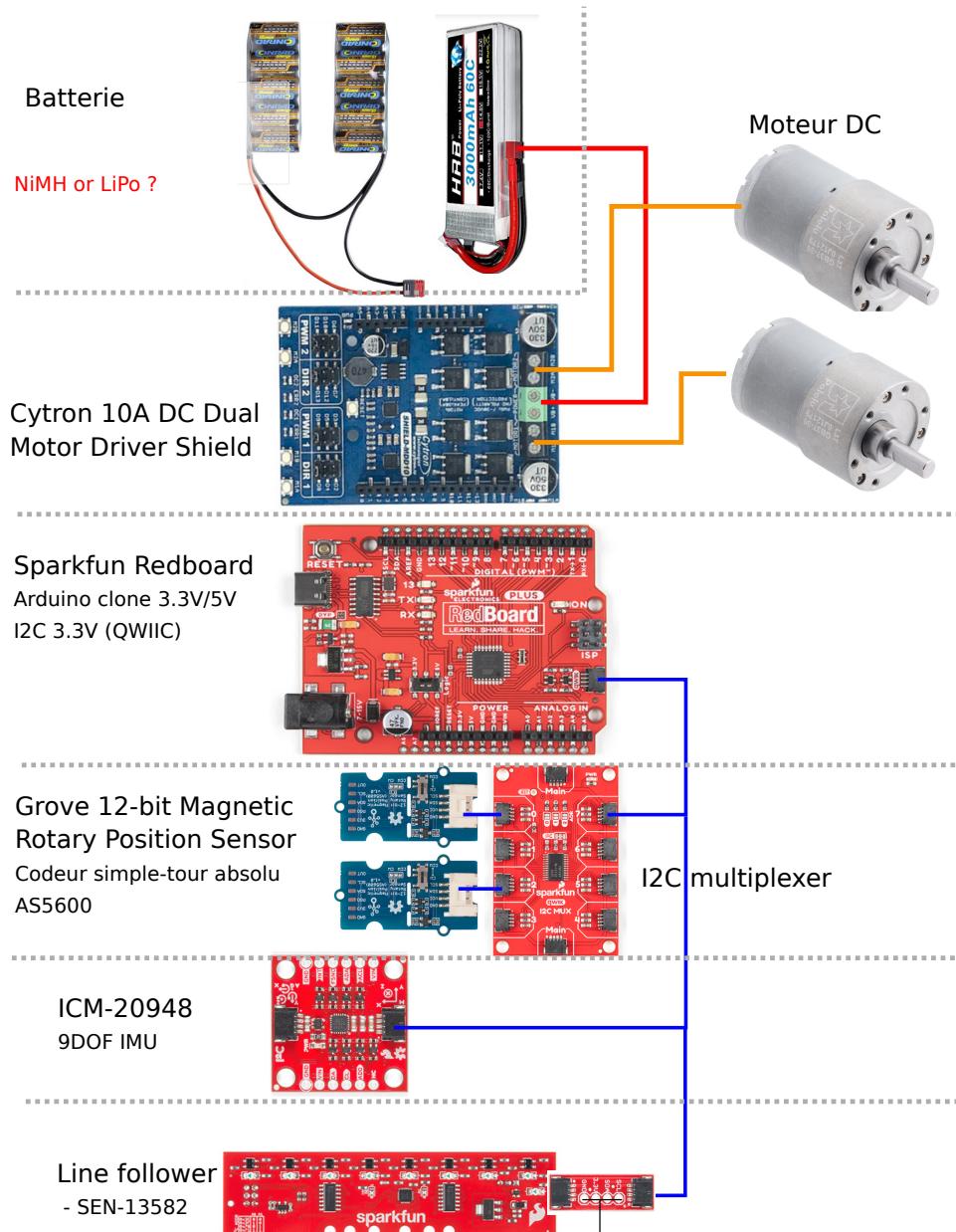


Figure 8: Plan de cablage général

Figure 7: Plan de cablage général