**Découvrez les principes mécaniques des robots autonomes**

Quand on entend parler de la carte Arduino, on imagine tout de suite bien des possibilités, en particulier la création de robots. Et bien on n’a pas tort, la carte Arduino permet en effet de programmer des robots autonomes !

Vous verrez dans ce chapitre ce que l’on entend par robot autonome et ce qui peut être imaginé en fonction de vos projets. Vous ferez aussi et surtout le point sur les notions de base de mécanique : transmission du mouvement, énergie, déplacement…

Mais commençons par le commencement !

**Au début il y avait l’homme, puis vint la machine**

Nous rêvons (moi en tout cas) de robots amis, qui nous seconderaient dans nos tâches quotidiennes, seraient subtiles et forts, intelligents et obéissants. La science-fiction et les films nous montrent d’ailleurs souvent ces robots. Bien que deux courants s’opposent avec les gentils robots (dans Interstellar, Wall-e, A.I. Intelligence Articielle…) et les méchants robots (dans Terminator, Matrix, 2001...), ils sont tous dotés d’intelligence artificielle, c'est-à-dire qu'ils réfléchissent par eux-mêmes et prennent leur propres décisions (comme le ferait un être humain). Je vais peut-être détruire l’un de vos plus grands rêves, mais ce type d’intelligence n’existe pas encore en robotique et en informatique.

En revanche, on peut amener une machine à agir et réagir avec son environnement, jusqu’à un point d’autonomie impressionnant (surveillance des centrales nucléaires, robots sur Mars, sondes…). Ces machines sont autonomes mais restent toutes obéissantes au programme qui les dirige.   
Il faut donc savoir que votre robot ne sera autonome (c'est-à-dire qu'il répondra seul aux sollicitations de son environnement) que si votre programme et les capteurs que vous lui avez fournis le lui permettent. Bref, il ne vous surprendra jamais, vous décevra souvent, et vous émerveillera parfois ! https://openclassrooms.com/bundles/common/images/smiley/hihi.png

Dans ce cours, lorsque nous allons parler de robots autonomes, cela se résumera à des robots qui savent se déplacer dans un environnement spécifique connu et qui réagissent à cet environnement selon les objectifs précis qui leur ont été donnés. Voici quelques exemples de robots autonomes qu’il est possible de réaliser avec la carte Arduino :

* Un portail automatique (si si, c’est une forme de robot) ;
* Un trieur de bille de couleur (oui, oui, ça aussi) ;
* Un dessinateur au feutre (imprimante simple) ;
* Un robot suiveur de ligne noire ;
* Un robot éviteur d’obstacle (type robot aspirateur) ;
* etc.

L’ensemble de ces robots ont en commun deux points importants : un mécanisme de mobilité et des capteurs.  
Je vais donc aborder ces deux points l’un après l’autre.

(Oui Lukas, vous pouvez entrer. Oui vous êtes en retard et vous venez de manquer mon introduction passionnante, mais je suis content de vous retrouver ! Installez-vous...)

Je vais commencer par un point bassement matériel...

**Quels matériaux pour un robot ?**

Souvent la difficulté est de construire son robot. La carcasse doit pouvoir contenir l'ensemble du matériel (carte Arduino, batteries, fils, moteurs, servos, capteurs) et être assez solide et légère à la fois pour une utilisation optimale.

Alors malheureusement je n'ai pas de recette miracle, et c'est presque tant mieux ! Il existe des dizaines de façons de concevoir son robot ou sa machine automatisée. Je vais vous donner quelques pistes tout de même...

**Le bois**

Plus précisément le bois de [balsa](https://fr.wikipedia.org/wiki/Balsa), offre une bonne alternative entre solidité et légèreté. Il se travaille au cutter, à la perceuse, au pyrograveur. Il s'assemble avec de la colle, des écrous et boulons, des vis à bois. On peut assez facilement concevoir sa carcasse robotique avec du balsa. Pour les projets de plus grande envergure, le contreplaqué est aussi un bon compromis (bien que plus lourd).

Un jouet en balsa (http://www.harborfreight.com)

L'inconvénient est qu'il faut être assez manuel et avoir les outils adaptés pour obtenir une réalisation correcte. le balsa est très utilisé en aéromodélisme pour sa légèreté.

Si vous vous sentez l'âme xylophile, n'hésitez pas ! Le coût reste assez faible (de l'ordre de 3€ la planche de 1 mm x 10 cm x 1m).

(J'ai dit xylophone, pas xylophone Lukas...)

**Le métal**

Il est bien plus solide, mais plus délicat à travailler. Souvent plus lourd (même si on utilise de l'aluminium), il permet d'obtenir une plus grande résistance aux torsions. Il existe plusieurs possibilités pour utiliser le métal. Soit acheter les plaques et les travailler soi-même, soit acheter des kits à base de tiges trouées et de visseries.

Un kit de construction en métal (www.adafruit.com)

Outre la difficulté technique pour ouvrager les pièces, le prix d'achat de ces kits est souvent assez élevé (environ 200€ pour le kit de l'image ci-dessus).

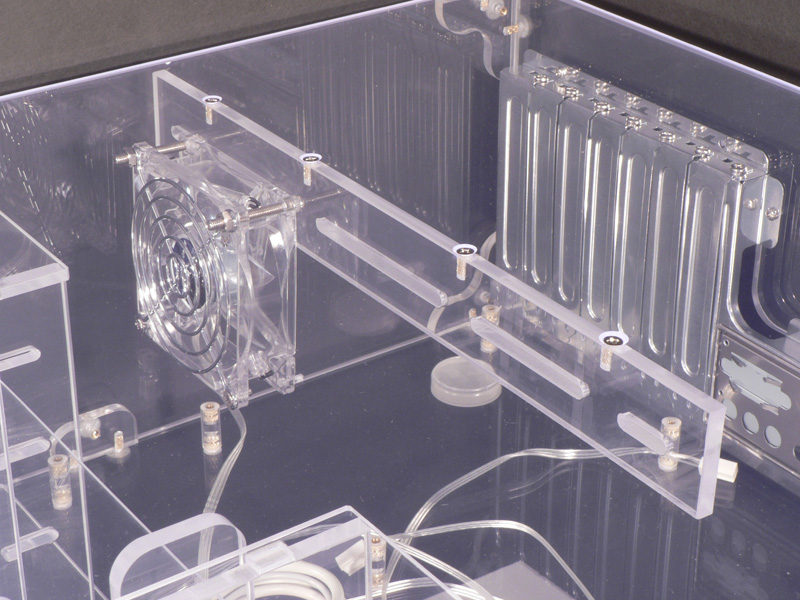
Maintenant, si vous êtes du métier ou que vous avez les ressources, les robots en métal sont souvent solides et esthétiques !

**Le plastique**

Alors il existe plusieurs sortes de plastiques :

* le plastique que vous recyclez de vos emballages divers ;
* le plastique acheté en plaque (type plexiglass) ;
* Le plastique moulé.

Là encore, le prix n'est pas toujours abordable (sauf dans le cas d’emballages recyclés) et outre ses qualités esthétiques, la production et la fixation des pièces nécessite un matériel adapté (assez proche du travail du bois).

Un boitier ordinateur en plexiglass (www.adafruit.com)

L'intérêt du plastique est, entre autre, sa transparence, qui permet des jeux de lumière et de couleur.

J’ai déjà eu l’occasion de voir des réalisations de robots entièrement en plastique de récupération qui ne palissaient pas devant des montages thermoformés... La patience, l'inventivité et l'habileté font souvent des merveilles !

**Les kits tout prêts**

Il existe dans le commerce des kits robots tout prêts, c'est-à-dire que vous avez tout ce qu'il faut pour construire votre robot roulant (carcasse et moteurs, ou carcasse permettant d'accueillir des moteurs et capteurs). C'est très pratique.

Une exemple de kit robot proposé par Arduino (www.geek.com)

Ces plateformes préfabriqués sont faciles à utiliser et répondent très souvent à vos besoins. Il en existe des dizaines avec des tailles et des formes différentes. Un conseil tout de même si vous investissez : regardez comment se fixe chaque module (carte Arduino, capteurs, moteurs, batterie) et surtout si vous pouvez détourner l'objet pour vos projets (ajout facile de servos, de capteurs...)

Le coût de ces plateformes varie de quelques euros à quelques centaines d'euro. En ce qui concerne les moteurs, préférez des moteurs qui ont une démultiplication intégrée, plutôt que des engrenages extérieurs (qui s'abîment plus vite çar moins protégés).

**Une solution intermédiaire : les LEGO™**

Alors ça peut faire rire, mais c'est la solution que je préfère ! Oui, à mon âge je joue encore aux LEGO™ !

(Vous aussi Lukas ? j'en suis ravi https://openclassrooms.com/bundles/common/images/smiley/triste.png)

Lorsque je dis LEGO™, je veux surtout parler des LEGO Technic ™ ou des LEGO Mindstorm ™. Ils ont toutes les sortes de pièces dont on a besoin et se montent facilement. On peut y adapter sans difficulté un servo-moteur ou des capteurs (bon oui, il faut oser percer une ou deux briques, et alors ?) et ils sont solides et légers.

Un modèle de LEGO Technic™ (www.amazon.fr)

L'inconvénient ? Le prix ! Une boîte de LEGO Technic™ peut facilement monter aux alentours de 200€ ! Maintenant, si vous faites comme moi, en recherchant vos anciens LEGO™(pour ceux qui y jouaient), en parcourant quelques bric-à-brac et en surfant sur le Web, vous devriez pouvoir acquérir facilement les objets de vos convoitises !

Un petit point sur les LEGO Mindstorm™ : ils sont conçus justement pour la robotique. La nouvelle gamme propose des capteurs, des moteurs, des télécommandes et un microcontrôleur. Un peu plus difficile à contourner que les anciens modèles, ils restent complètement adaptés pour la micro-robotique !

Une belle pub pour les kits LEGO Mindstorm™ (http://www.bluemagic.club)

Alors oui, ça donne envie, mais ça coûte les yeux de la tête et surtout on s'éloigne de l'objectif de l'Arduino : être libre de connecter ce que l'on veut. C'est la raison pour laquelle que parlais de solution intermédiaire.

Pour ma part j'utilise les LEGO™ pour la carcasse et pour les moteurs. Mais ensuite j'utilise l'Arduino et les capteurs électroniques faits maison pour tout le reste. Un bon compromis de mon point de vue.

Il existe des sites où les pièces peuvent être commandées à l'unité, et c'est largement suffisant et bien moins onéreux.

Bon il existe sûrement bien d'autres possibilités et vous aurez l'occasion d'échanger à ce sujet sur le [forum](https://openclassrooms.com/forum/sujet/mooc-arduino-perfectionnement).

Nous avons vu de quoi un robot peut être fait, il est de temps de le mettre en mouvement à présent…

Je ne parle pas seulement de déplacement, mais bien de mobilité. Dans l’exemple du portail, votre portail bouge autour d’un axe, ou sur un rail, mais ne se met pas à courir la campagne !  
Les deux principaux composants qui vont nous permettre de faire bouger nos robots sont : le moteur et le servo-moteur. Vous allez donc voir comment les utiliser au mieux pour répondre à vos besoins.

Parlons tout d'abord de transmission de mouvement...

**Mettez un robot en mouvement**

Le principe est simple : comment transmettre le mouvement de rotation de l'axe d'un moteur, à une roue ?

Alors vous me direz, il suffit de mettre la roue directement sur l'axe du moteur ! C'est en effet une des possibilités. La limite est la puissance fournie par le moteur. Nous avons vu dans le [cours d'initiation à Arduino](https://openclassrooms.com/courses/programmez-vos-premiers-montages-avec-arduino) que le moteur peut gagner en puissance (et du même coup perdre en vitesse) grâce à un jeu d'engrenages (ou autre). La seconde limite est qu'un moteur entraîne une seule roue, mais dans le cas d'une voiture, un moteur entraîne deux voire quatre roues. Il faut donc transmettre ce mouvement !

Pour cela, nous allons voir tout de suite trois méthodes possibles (il en existe d'autres) :

* la transmission sur deux plans parallèles ;
* la transmission sur deux plans perpendiculaires ;
* la vis sans fin (plans perpendiculaires et démultiplication accrue).

**Transmission sur le même plan**

Il s'agit de faire passer le mouvement depuis l'axe d'un moteur vers l'axe d'une roue, les deux axes étant parallèles. Deux méthodes s'offrent à nous : le train d'engrenage ou la poulie. L'un comme l'autre peuvent démultiplier le mouvement (c'est-à-dire qu'une petite roue entraîne une grosse roue). Si le moteur entraîne le petit engrenage (ou poulie), qui entraîne le plus grand sur lequel est fixé notre roue, le moteur tournera plus vite que la roue. Sa puissance sera plus importante mais la vitesse du robot sera réduite.

Voici deux exemples en vidéo.

Dans le premier exemple, le moteur entraîne un petit engrenage qui en entraîne un plus grand. L'axe de ce dernier est le même que celui de la roue.

Dans le deuxième exemple, le moteur entraîne une petite poulie qui entraîne ensuite une plus grande. L'axe de la grande poulie est le même que la roue.

Vous pouvez observer le mouvement des deux engrenages ou  des deux poulies. À chaque fois, le pignon (engrenage ou poulie de petite taille) entraîne l'autre partie qui est reliée à la roue. Les vidéos montrent un banc de démonstration (pour observer le mécanisme), il faut ensuite adapter le principe au robot.

Lorsqu'il s'agit d'engrenages, les rotations s'inversent d'une roue à l'autre, donc la roue tourne dans le sens inverse du moteur. Pour les poulies, les rotations sont les mêmes, donc la roue tourne dans le même sens que le moteur.

**Transmission sur plan perpendiculaire**

Parfois, le moteur n'est pas dans le même axe que la roue. Il faut utiliser dans ce cas un engrenage spécial qui permet de transmettre le mouvement à un axe perpendiculaire.

On peut ainsi, avec un seul moteur, entraîner deux roues situées sur le même axe. Sur la vidéo, le moteur entraîne l'engrenage directement. Il est tout à fait possible d'imaginer que le moteur entraîne un arbre (un axe long) avec au bout l'engrenage en question.

**La vis sans fin**

Ce principe permet une plus grande démultiplication de la puissance du moteur. L'axe du moteur entraîne une vis dont les ailes sont assez larges pour entraîner les dents d'un engrenage.  Ce dernier tourne donc très lentement et entraîne la roue qui du coup gagne énormément en couple (puissance de rotation).

Là encore, il s'agit  d'un banc de démonstration. Le montage sur un robot nécessitera que l'axe de la vis et de l'engrenage soient fortement maintenus en place.

Il existe encore d'autres méthodes de transmission de mouvement, mais vous devriez pouvoir trouver votre bonheur parmi ces trois méthodes principales.

Imaginons maintenant que votre robot soit équipé de moteurs qui entraînent des roues (par démultiplication ou non). Nous allons voir maintenant comment faire pour que ce robot tourne et n'aille pas uniquement tout droit !

(Lukas, ça suffit maintenant avec les LEGO™! Le pire, ce sont vos bruitages ! Ça vous fait postillonner, c'est... https://openclassrooms.com/bundles/common/images/smiley/blink.gif ... bref.)

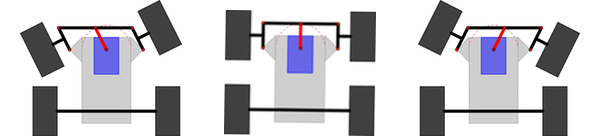
**Faites tourner votre robot**

Il y a deux écoles pour faire tourner une plateforme robotisée : l'utilisation de roues directrices (comme dans une voiture) ou l'utilisation de sens de rotation inversés des trains de roues (comme dans un tank). Nous aborderons aussi une méthode pour articuler un bras (qui est souvent liée à une rotation).

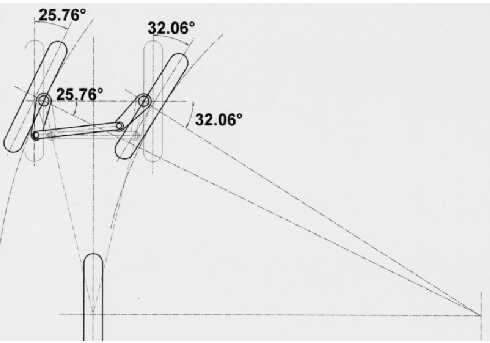
J'utiliserai le terme "plateforme" pour parler d'un ensemble de mécanismes qui sert au déplacement. J'utiliserai le terme "robot" lorsque cette plateforme sera pilotée par une technologie programmable, comme la carte Arduino. (c'est pour moi la différence entre voiture radio-commandée et robot autonome !)

**Les roues directrices (la voiture)**

Le principe consiste à ce que la plateforme qui avance dispose de roues qui vont s'orienter dans la direction désirée pendant le mouvement. Ces roues peuvent être situées à l'avant ou à l'arrière. La plateforme se déplace en faisant un arc de cercle dont le rayon est plus ou moins grand (le fameux rayon de braquage). Plus l'angle que prennent les roues directrices par rapport au sens du mouvement est grand, plus le rayon du cercle de rotation sera petit. La plateforme tournera sur une zone réduite. À l'inverse, moins l'angle pris par les roues directrice est grand, plus le rayon de braquage est important, et plus il faut d'espace pour faire tourner la plateforme.

Roues directrices

La barre rouge est souvent pilotée par un servo-moteur. Il est important que les axes soient alignés (en haut comme en bas) pour que la rotation s'effectue correctement. Cette construction ne prend pas en compte une différence entre la roue droite et la roue gauche pour l'angle de rotation. Pour un robot de petite taille non habité, ce n'est pas très grave, mais pour un véhicule plus lourd, roulant plus vite et surtout avec des humains à l'intérieur, il est nécessaire de prendre en compte cette différence. Voici un schéma explicatif :

Angle différents lors de la rotation des roues (http://moteurdiesel.voila.net)

Sur ce schéma, on comprend que les deux roues doivent être sur des tangentes de cercles de même centre. Le rayon de ces deux cercles et le positionnement des roues étant différents, on observe bien que pour que les roues suivent correctement la trajectoire, il faut une inclinaison différente (ici la différence est de presque 7°).

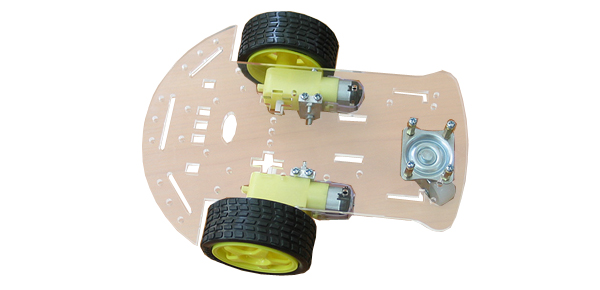
Mais je le répète, pour un robot, ce n'est pas primordial. ;)

Passons maintenant à la deuxième méthode possible pour faire tourner une plateforme.

**Rotation inversées des roues (le tank)**

On utilise ce mode pour des robots à roue folle ou robots à chenilles.

Le robot à roue folle, c'est celui qui a deux roues (une à droite, l'autre à gauche) qui peuvent tourner indépendamment l'une de l'autre (dans les deux sens) et une troisième roue dite folle, c'est-à-dire qui ne se contente que de suivre le mouvement (quel que soit le sens).

Plateforme à roue folle (http://www.robot-maker.com)

Les roues folles peuvent être faites un peu comme les roues libres des caddies de supermarché, ou avec une bille dans une sorte de rotule :

Roue bille à gauche et roue libre à droite (www.discount.com)

Les deux se fixent sous la plateforme et permettent à la fois de l'équilibrer et de la laisser libre de tourner.

Le robot à chenille, c'est le tank de base. Deux moteurs entraînent deux chenilles situées de part et d'autre de la plateforme :

Plateforme à chenilles (http://tpe-lamartin.e-monsite.com)

Dans les deux cas, le principe est le même :

* Les roues (ou chenilles) tournent dans le même sens, la plateforme avance ou recule.
* Une roue (ou chenille) est arrêtée, l'autre continue de tourner, la plateforme tourne sur un cercle dont le centre est le point de contact de la roue arrêtée avec le sol. C'est le virage lent.
* Une roue (ou chenille) tourne dans un sens et l'autre dans le sens contraire, la plateforme tourne sur un cercle dont le centre se situe entre les deux roues. C'est le virage rapide ou sur place.

En ce qui concerne les robots à chenilles, il faut savoir que les frottements au sol sont plus importants lors des rotations.

Ce type de plateforme est beaucoup plus pratique à gérer dans les cas ou les robots doivent pouvoir tourner rapidement (voire sur eux-mêmes). Si votre montage comprend un L293D (puce qui permet de gérer deux moteurs dans les deux sens), vous pouvez sans problème vous orienter vers ce type de construction.

Voyons maintenant un principe d'articulation entre deux pièces, celui du bras mécanique.

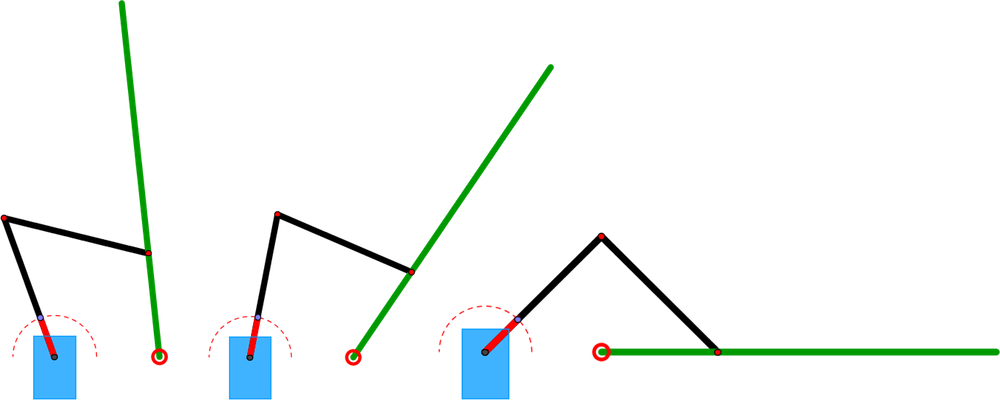
**Actionner un bras mécanique**

Ce type de construction concerne les pelleteuses, les portails à battants, les plans qui s'inclinent...

Nous allons prendre l'exemple du portail à battant (comme une porte quoi). Pour qu'il s'ouvre, vous avez deux possibilités :

* soit vous placez le servo ou le moteur directement sur son axe,
* soit vous utilisez le principe du bras.

Dans ce deuxième cas, le servo-moteur (ou le moteur) n'est pas situé sur l'axe, mais décalé, et un bras (composé de deux parties) est lié d'un côté au servo-moteur, de l'autre au portail (ou plan à déplacer). Dans le schéma qui suit, le servo (en bleu) est fixe. Son axe (rouge) est mobile, il entraîne la barre noire (qui le prolonge), qui entraîne la deuxième barre noire, qui elle-même entraîne la barre verte (le plan ou le portail).

Différentes positions du plan (en vert) en fonction de la position de l'axe du servo-moteur (partie fixe en bleu, partie mobile en rouge)

On voit bien sur le schéma que l'amplitude de la rotation de l'axe du servo (environ 45°) permet une amplitude plus grande pour le plan (environ 95°). À vous bien sûr de faire vos réglages pour que le montage corresponde à vos attentes !

Bien, je crois que nous en avons terminé avec les mouvements. Il en existe d'autres bien sûr, mais l'idée est toujours de vous proposer la base, au moins pour vos donner des idées.

Un dernier point avant de terminer ce chapitre, car pour que vos robots fassent tous ces mouvements, ils vont bien avoir besoin d’énergie…

**Alimentez votre robot en électricité**

S'il y a une chose à ne pas négliger dans le cas des robots autonomes, c'est bien la façon dont vous allez alimenter votre robot en électricité.

Lorsqu’on utilise des moteurs, il est important de séparer le circuit de commande du circuit de puissance (consultez ce [point sur les circuits](https://openclassrooms.com/courses/programmez-vos-premiers-montages-avec-arduino/le-moteur-a-courant-continu-partie-1-transistors-et-sorties-pwm#/id/r-3285327) de mon cours d’initiation à Arduino si vous avez besoin de vous rafraîchir la mémoire !).

Il vous faut donc prévoir deux circuits. Vous allez me dire :

"Pas de problème, il suffit d’alimenter l’Arduino d’un côté et les moteurs de l’autre en séparant les deux circuits mais en utilisant la même source ! En gros, une pile qui alimente en parallèle mon Arduino et mes moteurs."

Et bien je vous déconseille cette méthode tout simplement pour des questions de consommation. Si vos moteurs (au démarrage entre autre) puisent trop de courant, vous risquez d'en manquer pour l'Arduino et du coup perdre le contrôle du robot (si l'Arduino s'arrête, même momentanément). Et puis, même si l'Arduino peut fonctionner en recevant des tensions jusqu'à 12V, une pile de 9V suffit. Ensuite à vous de choisir et de chercher les générateurs qui conviendront le mieux à vos montages.

Je ne vais pas m'étendre sur les types de piles, générateurs ou accumulateurs électriques (de leur petit nom “accus”). Sachez juste que vous pouvez faire fabriquer le modèle qui vous sied parfaitement dans des magasins spécialisés (c'est souvent un peu plus cher, mais le résultat n'est pas négligeable).

Les questions à vous poser pour l'alimentation de votre robot sont les suivantes :

* Où vais-je mettre mes piles ou batteries sur ma plateforme ?
* Quelles sont l'ampérage et la tension dont j'ai besoin ?
* Quel type de batterie utiliser (poids, rechargeables ou non) ?
* Ai-je bien séparé l'alimentation du circuit de commande (Arduino) de celle du circuit de puissance (moteurs, servos...) ?

Un dernier conseil : il est assez pratique de prévoir pour chaque alimentation un interrupteur manuel. Il permet de couper ou non les alimentations pour lancer le robot. C'est bien mieux pour les piles, et ça permet de ne pas avoir un robot qui roule tout le temps, même quand on le transporte. ;)

**En résumé**

Vous avez appris (ou revu) dans ce chapitre :

* Quels matériaux peuvent être utilisés pour la fabrication de vos plateformes robotisées  (balsa, métal, plastique...) ;
* Comment transmettre un mouvement entre le moteur (ou le servo-moteur) et une roue ;
* Comment faire tourner une plateforme mobile (comme un tank ou une voiture) et une articulation ;
* Quels sont les points à ne pas oublier pour alimenter votre robot en électricité.

Cette fois, je pense qu’on a fait le tour sur les bases de la mécanique des robots autonomes. Voyons comment utiliser des capteurs avec ces robots…

**Faites réagir votre robot avec des capteurs**

Maintenant que votre robot est mobile, il va devenir important de savoir ce qu'il doit faire.

Un robot doit avoir une mission (même si elle paraît très simple) car c'est ce qui va guider votre construction, le choix de vos capteurs et votre programmation.

Nous avons vu dans le cours d’initiation à Arduino, plusieurs [capteurs](https://openclassrooms.com/courses/programmez-vos-premiers-montages-avec-arduino/les-capteurs-electroniques) et la façon de les connecter et de les programmer.

Nous allons dans ce chapitre observer ces capteurs sous un autre angle : quel capteur va répondre à vos attentes ?

Si vous voulez réaliser ce chapitre en faisant tous les tests, il vous faut prévoir :

* Deux microrupteurs
* Une LED infrarouge et une photodiode (capteur infrarouge)
* Un capteur ultrasons SRF05

Vous trouverez le détail des connexions de ces capteurs dans les annexes du cours d'initiation à l'Arduino

Je vais donc aborder plusieurs types de missions et voir comment équiper le robot en conséquence !

Bien sûr, un robot peut tenter de cumuler l'ensemble des missions, mais vous verrez que du coup, ce n'est pas si évident.

J'oriente ce chapitre sur un robot autonome mobile, avec une roue folle. Il dispose donc pour le moment de deux moteurs, de la carte Arduino (alimentée séparément) et donc de la possibilité d'avancer, de reculer, de tourner sur place (le tout plus ou moins rapidement).

Attaquons notre première mission...

Je vais parler du robot en utilisant parfois la première personne du singulier (c'est "je" pour votre information Lukas) tout simplement parce que se mettre à la place du robot permet parfois de mieux imaginer le programme à effectuer.

**Je contourne un obstacle que je touche**

Alors pour éviter un obstacle que l'on touche, et bien il faut déjà se rendre compte qu’on l’a touché ! Il vous faut donc des capteurs de contact de type tout ou rien : soit je touche un obstacle, soit je ne le touche pas.

Le principe est simple : si le robot avance il doit pouvoir se rendre compte qu'il touche un obstacle.

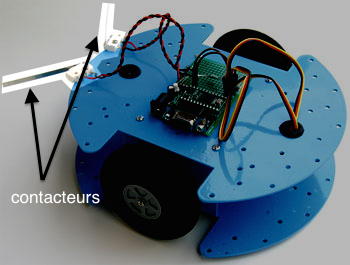
Les microrupteurs sont tout indiqués pour cette mission. Mais où les placer ?

Je dirais que c'est le plus délicat ! En effet, en fonction de la forme de votre robot ou de l'obstacle, le point de contact n'est pas toujours au même endroit.

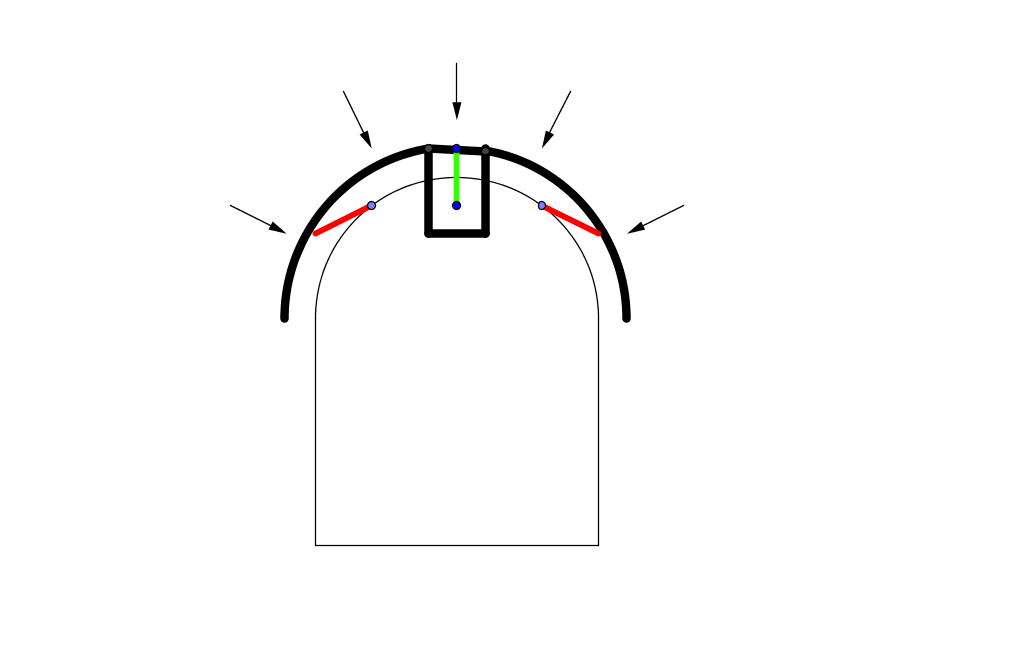
**Le modèle "moustaches de chat"**

Nous allons imiter le principe des moustaches qui servent au chat à détecter des objets. En effet si quelque chose touche la moustache du chat, il le sent (Non Lukas,  nous n’avons pas besoin de votre chat pour vérifier). De même, si le contact est à gauche ou à droite, il sait où se trouve l'obstacle.

Il vous suffit donc de placer des microrupteurs à l'avant de votre robot. Il vous en faudra au moins deux (pour différencier l'obstacle à droite et l'obstacle à gauche). Voici une plateforme qui montre un tel montage :

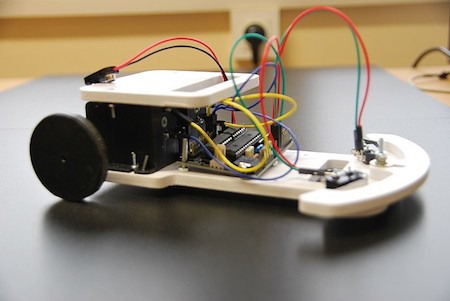
Les contacteurs à l'avant du robot agissent comme des moustaches (www.elektormagazine.fr)

La limite de ce système est que si l'obstacle est fin et arrive exactement entre les deux moustaches, le robot ne va pas savoir qu'il est bloqué. On peut ruser en ajoutant une barre souple d'une moustache à l'autre qui, du coup, couvrira toute la zone. Il existe une petite construction qui permet d'affiner ce montage : le pare-choc.

Le pare-choc schématisé

La pièce noire est mobile verticalement sur l'axe vert et peut aussi pivoter à droite ou à gauche (point de pivot en bas de l'axe vert). Du coup, la zone couverte s'étend sur tout l'avant sur presque 180°. Le plus difficile est de concevoir une pièce qui déclenche bien les contacts en fonction de la zone touchée (contacteur gauche pour la gauche, contacteur droite pour la droite et les deux pour le centre) et qui revient à sa place ensuite. Il peut donc être utile d'ajouter de petits ressorts (au niveau de l'axe vert et des côtés) pour que la pièce reprenne sa position.

Voici un exemple :

Un pare-choc avant qui différencie droite, gauche et centre (http://www.clg-rondeau-rambouillet.ac-versailles.fr)

Ce pare-choc n'est pas semi-circulaire, mais il suffit à répondre à l'objectif : différencier droite, gauche et centre.

Notez que les roues ne dépassent pas de la zone du pare-choc. C'est important.

Comment coder la réaction du robot ?

Votre programme doit être organisé correctement pour que ce soit clair. Il vous faut donc :

* Créer des fonctions pour avancer et reculer (2 moteurs dans le même sens), tourner à droite ou à gauche (1 moteur dans un sens et l'autre en sens inverse, ou 1 moteur arrêté et l'autre en marche), et s'arrêter (2 moteurs à l'arrêt).
* Connecter correctement vos microrupteurs pour repérer un contact éventuel, et  les repérer  par des noms de variable compréhensibles (ex: contactGauche et contactDroite).

Une fois que c’est fait, et bien le reste est assez simple... il suffit de penser à la place du robot :

* Si je sens un contact à droite, c'est que l'obstacle est à droite. Je m'arrête, je recule pendant un temps donné, je tourne à gauche pendant un temps donné, puis j'avance à nouveau.
* Si je sens un contact à gauche, c'est que l'obstacle est à gauche. Je m'arrête, je recule pendant un temps donné, je tourne à droite pendant un temps donné, puis j'avance à nouveau.
* Si je sens un contact à droite et à gauche, c'est que l'obstacle est devant. Je m'arrête, je recule pendant un temps donné, je tourne à droite (ou à gauche) pendant un temps donné, puis j'avance à nouveau.

C'est tout ! Du coup, si vos fonctions sont claires, le code sera très lisible ! Voici un exemple en pseudo code :

//initialisation des variables

//on définit les pin pour chaque contact

int contactGauche = 3; //microrupteur connecté au pin 3 en mode pull-up

int contactDroite = 4; //microrupteur connecté au pin 4 en mode pull-up

void setup() {

pinMode(contactGauche, INPUT\_PULLUP); //pas d'utilisation de résistance externe

pinMode(contactDroite, INPUT\_PULLUP); //idem

avance();

}

void loop() {

// les pins mode INPUT\_PULLUP, on va donc les lire en inversant leur valeur pour que les tests soient plus clairs

boolean toucheGauche = !digitalRead(contactGauche); //on créé un variable d'état en inversant la valeur du pin

boolean toucheDroite = !digitalRead(contactDroite); //on créé une autre variable en inverant aussi sa valeur

if (toucheGauche && toucheDroite) { // contact au centre

arret();

delay(500);

recule();

delay(1000);

tourneDroite();

delay(1000);

avance();

}

else if (toucheGauche && !toucheDroite) { // contact à gauche

arret();

delay(500);

recule();

delay(1000);

tourneDroite();

delay(1000);

avance();

}

else if (!toucheGauche && toucheDroite) { // contact à droite

arret();

delay(500);

recule();

delay(1000);

tourneGauche();

delay(1000);

avance();

}

delay(100); //attente pour bien repérer le contact au centre éventuel

}

//fonctions de mouvement de la plateforme

void arret() {

//commande d'arrêt des deux moteurs

}

void avance() {

//commande pour faire tourner les deux moteurs dans le même sens

}

void recule() {

//commande pour faire tourner les deux moteurs dans le même sens mais en arrière

}

void tourneDroite() {

//commande pour faire tourner un moteur dans un sens et l'autre dans le sens inverse afin que la plate-forme tourne à droite

}

void tourneGauche() {

//commande pour faire tourner un moteur dans un sens et l'autre dans le sens inverse afin que la plate-forme tourne à gauche

}

Ce programme vous semble complètement indéchiffrable ? Il vous manque encore quelques bases de programmation, je vous conseille de reprendre mon [cours d’initiation à Arduino](https://openclassrooms.com/courses/programmez-vos-premiers-montages-avec-arduino) pour y voir plus clair.

Notez que je n’ai pas écrit le programme de chaque fonction dirigeant le mouvement du robot (avance(),  recule(), etc.), l’objet de ce chapitre étant plutôt de ce concentrer sur le programme global et l’interaction avec les capteurs. Vous pouvez les construire en vous référant au cours d'initiation ([chapitre sur les moteurs](https://openclassrooms.com/courses/programmez-vos-premiers-montages-avec-arduino/le-moteur-a-courant-continu-partie-2-le-pont-en-h-et-les-circuits-integres)) avec l'utilisation d'un L293D, ou bien voir comment créer de telles fonctions avec un shield moteur dans un prochain chapitre !

Je n'ai pas défini de fonction  stop() car c'est un mot-clé du langage Arduino (on le voit par la coloration syntaxique), j'ai donc renommé ma fonction  arret().

N'oubliez pas qu'il ne faut pas d'accent dans les noms de variables, constantes ou fonctions. ;)

Vous pouvez tester sur votre plateforme, le robot va agir comme sa mission le lui demande ! C'est-y-pas-merveilleux ?!

Si vous n'avez pas de plateforme encore construite ou si vous souhaitez vérifier le bon fonctionnement de votre code, voici un moyen de suivre le déroulement du programme. Pour le montage, vous connectez deux boutons poussoir sur les pins 3 et 4 de l'Arduino en mode pull-up. En appuyant sur celui de droite, celui de gauche ou les deux, vous devriez obtenir la bonne séquence d'affichage sur la console série.

//initialisation des variables

//on définit les pin pour chaque contact

int contactGauche = 3; //microrupteur connecté au pin 3 en mode pull-up

int contactDroite = 4; //microrupteur connecté au pin 4 en mode pull-up

void setup() {

Serial.begin(9600);

pinMode(contactGauche, INPUT\_PULLUP); //pas d'utilisation de résistance externe

pinMode(contactDroite, INPUT\_PULLUP); //idem

avance();

}

void loop() {

// les pins mode INPUT\_PULLUP, on va donc les lire en inversant leur valeur pour que les tests soient plus clairs

boolean toucheGauche = !digitalRead(contactGauche); //on créé une variable d'état en inversant la valeur du pin

boolean toucheDroite = !digitalRead(contactDroite); //on créé une autre variable en inversant aussi sa valeur

if (toucheGauche && toucheDroite) { // contact au centre

Serial.println("Je touche au centre");

arret();

delay(500);

recule();

delay(1000);

tourneDroite();

delay(1000);

avance();

}

else if (toucheGauche && !toucheDroite) { // contact à gauche

Serial.println("Je touche a gauche");

arret();

delay(500);

recule();

delay(1000);

tourneDroite();

delay(1000);

avance();

}

else if (!toucheGauche && toucheDroite) { // contact à droite

Serial.println("Je touche a droite");

arret();

delay(500);

recule();

delay(1000);

tourneGauche();

delay(1000);

avance();

}

delay(100);

}

//fonctions de mouvement de la plateforme

void arret() {

Serial.println("Je m'arrete");

//commande d'arrêt des deux moteurs

}

void avance() {

Serial.println("J'avance");

//commande pour faire tourner les deux moteurs dans le même sens

}

void recule() {

Serial.println("Je recule");

//commande pour faire tourner les deux moteurs dans le même sens mais en arrière

}

void tourneDroite() {

Serial.println("Je tourne a droite");

//commande pour faire tourner un moteur dans un sens et l'autre dans le sens inverse afin que la plate-forme tourne à droite

}

void tourneGauche() {

Serial.println("Je tourne a gauche");

//commande pour faire tourner un moteur dans un sens et l'autre dans le sens inverse afin que la plate-forme tourne à gauche

}

Ce genre de code vous permet de vérifier (ou de débugger) votre programme sans avoir à tout connecter.

Ne connectez rien sur les pins 0 et 1 de l'Arduino si vous utilisez la console ! Ils servent justement pour la communication avec votre ordinateur.

Vous remarquerez que je n’ai pas mis d’accents dans les messages console (ex : Serial.println(“Je tourne a droite”)). Prenez-en l’habitude, ces messages sont codés en ASCII qui a été inventé par les anglophones, donc qui ne contient pas d’accent !

Ce modèle “moustaches de chat” a-t-il des limites ?

(Je vois que vous suivez Cunégonde, toujours la petite question qui pique https://openclassrooms.com/bundles/common/images/smiley/hihi.png)

Oui évidemment ! Je vous dresse une petite liste des problèmes probables :

* L'obstacle est en hauteur ou trop bas pour toucher les moustaches ou le pare-choc. Vous voyez alors votre robot rentrer bêtement dedans et s'obstiner à avancer, quitte à arracher tout son matériel (fils et autres), et vous vous précipiterez pour le tirer de là (ha les parents !).
* Votre robot évolue sur un plan en hauteur (genre table du salon). Il ne va pas détecter le vide qui s'approche et qui le menace. Si vous ne réagissez pas assez vite, il va tomber (bêtement encore) et sûrement se casser un truc ou pire...
* Il est programmé pour que lorsqu'il touche un objet, il recule avant de tourner. En théorie, il retourne d'où il est passé sans encombre. Mais si votre chien, chat, petite soeur passe à ce moment, le robot le/la percutera (oui Lukas, ça marche aussi avec une grand-mère) ce qui provoquera une réaction improbable du sujet percuté. Le robot lui continuera son processus (j'allais dire bêtement ;) ).

C'est la raison pour laquelle les robots doivent avoir des missions précises dans un environnement précis. C'est plus sûr pour tout le monde !

Voyons maintenant une mission un peu plus difficile...

**J'évite les obstacles sans les toucher**

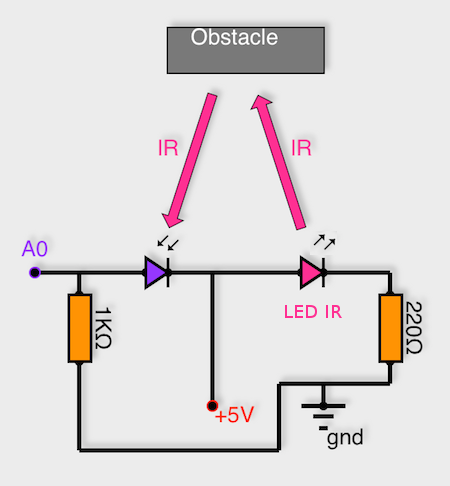
Voilà qui est plus ardu. Sans toucher veut dire que l'on doit pouvoir détecter un obstacle à distance et évaluer à quelle distance on se trouve de l'obstacle. Nous avons deux sortes de capteurs qui peuvent répondre à nos besoins. Il fonctionnent sur le même principe : capter un écho.

* Le premier est le capteur infrarouge, qui va capter la lumière infrarouge qui rebondit sur l'obstacle.
* Le second est le capteur à ultrasons, qui va capter un son qui rebondit sur un obstacle.

**Le capteur de distance à infrarouge**

Nous avons vu dans le cours d'initiation comment utiliser une [photodiode](https://openclassrooms.com/courses/programmez-vos-premiers-montages-avec-arduino/les-capteurs-electroniques#/id/r-3309355) (ou diode réceptrice infrarouge), voyons maintenant comment l'utiliser comme capteur de distance.

Il nous faut réaliser un montage qui envoie de la lumière infrarouge et qui la capte, dans le même temps. Voici le schéma de principe :

Principe de la détection de distance par réflexion de lumière infrarouge

Vous remarquerez que j'ai choisi le montage de la photodiode "à l'envers" (voir cours d'initiation). L'autre convenait tout autant.

L'idéal est de prévoir une petite séparation entre la diode émettrice et la réceptrice, pour éviter qu'elles ne se "voient" directement (entourer la photodiode d'un petite tube en plastique noir genre paille est largement suffisant).

Voici un capteur de distance infrarouge tout fait :

Un capteur de distance infrarouge du commerce (www.robotshop.com)

Plus l'obstacle va être proche, plus la quantité de lumière infrarouge renvoyée par l'obstacle va être importante, donc plus la tension captée et numérisée par le CAN A0 va avoir une valeur élevée. Pour résumer, plus la valeur captée par A0 est élevée, plus l'objet est proche. Il suffit donc ensuite de programmer le robot pour qu'il agisse en fonction de la distance de l'objet. Nous allons le voir un peu plus loin.

Ça marche sur toutes sortes d'obstacles ? Parce que je croyais que le noir absorbait les ondes lumineuses ?

(Ha sacré Cunégonde ! Vous ne lâchez rien hein ?)

Alors en effet, il y a deux principaux inconvénients avec ce capteur :

* La distance mesurable reste faible (environ 10 cm si le capteur est bien conçu) ce qui limite la vitesse de votre robot. En effet s'il arrive trop vite sur l'obstacle et que le capteur n'a pas eu le temps d'analyser la distance, vous risquez le choc.
* La couleur de l'obstacle agit sur la distance mesurable. En effet, un objet de couleur noire ne va quasiment rien réfléchir, donc ne sera pas détecté par votre capteur (Lukas, rassurez-vous, même si vous ne réfléchissez pas souvent, vous seriez vus par le capteur).

Vous pouvez améliorer la réception de la réflexion en mettant plusieurs diode IR afin d'émettre plus de lumière infrarouge.

Le cône de réception (la zone de détection) fait que votre robot pourra capter la présence d'un objet légèrement au-dessus ou en-dessous du capteur. C'est donc un peu mieux qu'une moustache. Mais pour une détection optimale, il faudrait presque trois capteurs (devant, devant-droite et devant-gauche) donc trois pins analogiques utilisés...

Évidemment, pour trouver le placement le plus efficace pour votre capteur, il vous faudra faire plusieurs tests et ajustements.

Avant de voir le code type, passons au capteur à ultrasons.

**Le capteur de distance à ultrasons**

Nous avons abordé dans le cours d'initiation ce [capteur (SRF05)](https://openclassrooms.com/courses/programmez-vos-premiers-montages-avec-arduino/les-capteurs-electroniques#/id/r-3309370) et les calculs de distances qui y sont liés. C'est d'ailleurs la grande différence entre le capteur de distance à  infrarouge (dont il est assez difficile de paramétrer une mesure de distance précise) et le capteur à ultrasons qui permet presque d'obtenir une mesure au centimètre (à condition que l'obstacle se situe entre 5cm et 150 cm, au-delà de ces limites la fiabilité diminue).

Le cône de détection est aussi plus important que le capteur infrarouge (entre 30 et 60° vers la zone concernée). Le robot détecte donc des objets plus haut, plus à droite, plus à gauche et plus bas.

Plusieurs méthodes de programmation sont possibles :

* Soit on procède comme nous allons le voir ensuite, c'est-à-dire en réalisant une mesure régulière.
* Soit on procède comme la chauve-souris (non Lukas, on ne dort pas la tête en bas) c'est-à-dire que plus on capte que l'objet se rapproche, plus on fait des mesures rapprochées dans le temps, et plus il s'éloigne, plus on espace les mesures. Cette méthode libère du temps processeur mais risque de ne pas détecter un objet qui surgirait dans le champs. Mais comme nos robots évoluent dans un milieu protégé...

Il est tout à fait possible de monter le capteur sur un servo-moteur, afin qu'il scanne de droite à gauche pendant qu'il roule en ligne droite. Ou bien, lors de l'approche d'un obstacle, plutôt que faire tourner le robot, on fait juste tourner la "tourelle" du capteur pour chercher le meilleur chemin.

Voyons maintenant le programme type permettant au robot d’éviter un obstacle, qu’il l’ait détecté à distance avec un capteur à infrarouge ou à ultrasons.

**Programme d'évitement d'obstacle sans contact**

La structure de ce programme est globalement la même pour le capteur infrarouge et pour le capteur à  ultrasons. Voici le principe :

* Je roule tout droit (en mesurant)  tant que tout va bien.
* Je détecte que je m'approche d'un obstacle.
* Je ralentis (facultatif, mais permet d'observer si l'obstacle persiste ou s'il disparaît).
* Je détecte que l'objet est trop près de moi, je m'arrête.
* Je recule un peu (facultatif, mais permet de ne pas toucher pendant les rotations).
* Je tourne sur moi-même (ou fais pivoter ma tourelle) à droite puis à gauche pour chercher un passage (mesure d'une distance assez grande).
* Là, deux possibilités de programme : Soit je m'engage dans le passage dès que je le trouve, soit je m'engage dans le passage le "plus libre" (ce qui nécessite de stocker les mesures effectuées en fonction de la position, puis de s'orienter correctement).
* Je reprends ma route.

Il vous faut donc créer des fonctions de mouvement (on peut ajouter une notion de vitesse en paramètre) comme pour les moustaches. Il est toujours préférable de créer une fonction de mesure de distance (qui renvoie soit une distance, soit un état, par exemple : ok, proche, stop).

Vous trouverez les principes de ces fonctions dans le chapitre [des moteurs](https://openclassrooms.com/courses/programmez-vos-premiers-montages-avec-arduino/le-moteur-a-courant-continu-partie-1-transistors-et-sorties-pwm) sur le cours d'initiation ou dans la section suivante si vous utilisez un shield moteur.

Voici un exemple de code. Il utilise le capteur SRF05. J'ai ajouté la communication avec le moniteur série pour les tests, elle est bien sûr facultative (mais bien pratique pour tester son programme ;)) :

#define PROCHE 20

#define DANGER 10

#define AFOND 200

#define PRUDENT 50

#define NBMESURE 5

#define VSON 59

//initialisation

int pinSRF=2; //pin numérique utilisé pour le capteur ultrasons.

void setup() {

Serial.begin(9600);

avance(AFOND); //on démarre le robot

}

void loop() {

switch (mesure()){ //utilisation de la condition switch

case 0: //l'obstacle est dans la zone DANGER

Serial.println("\*DANGER\*");

arret(); // on arrête le robot

delay(200);

recule(PRUDENT); // on le fait reculer un peu

delay(200);

recherche(); // on recherche la bonne voie

break;

case 1: //l'obstacle est dans la zone PROCHE

Serial.println("Attention...");

avance(PRUDENT); // on ralentit la vitesse

default:

avance(AFOND);

}

}

//fonctions

int mesure(){

//fonction qui mesure une distance avec le capteur

unsigned long mesure = 0; // variable de mesure

unsigned long cumul = 0; //variable pour la moyenne

for (int t = 0; t < NBMESURE; t++) { // boucle pour effectuer les mesures

pinMode (pinSRF, OUTPUT); //on prépare le pin pour envoyer le signal

digitalWrite(pinSRF, LOW); //on commence à l'état bas

delayMicroseconds(2); //on attend que le signal soit clair

digitalWrite(pinSRF, HIGH);//mise à l'état haut

delayMicroseconds(10); //pendant 10 µs

digitalWrite(pinSRF, LOW); //mise à l'état bas

pinMode(pinSRF, INPUT); //on prépare le pin pour recevoir un état

mesure = pulseIn(pinSRF, HIGH); // fonction pulseIn qui attend un état haut et renvoie le temps d'attente

cumul+=mesure; //on cumule les mesures

delay(50); //attente obligatoire entre deux mesures

}

mesure=cumul/NBMESURE; //on calcule la moyenne des mesures

mesure=mesure/VSON;//on transforme en cm

if (mesure<=DANGER){//on teste si l'obstacle est dans la zone DANGER

return 0; //si oui, on retourne le code de danger

}

else if (mesure>DANGER && mesure<=PROCHE){//on teste s'il est dans la zone PROCHE

return 1; //si oui, on retourne le code de proche

}

return 2; // on retourne le code de sans risque

}

void recherche(){

//fonction pour rechercher un passage

tourneGauche(AFOND); //on positionne le robot à gauche (facultatif)

delay (300);

int etat=0; //variable de test de possibilité d'avancer

while (etat!=2){ // tant que la distance n'est pas suffisante

etat=mesure(); // on effectue la mesure

tourneDroite(PRUDENT); //on fait tourner le robot

}

Serial.println("La voie est libre !");

//retour au programme principal

}

void arret(){

Serial.println("Je stoppe");

//fonction d'arrêt des moteurs

}

void avance(int v){

Serial.print("J'avance");

affiche(v);

//fonctin de mise en route des deux moteurs dans le sens avant

// on utilise la variable v pour le pilotage PWM

}

void recule(int v){

Serial.print("Je recule");

affiche(v);

//fonctin de mise en route des deux moteurs dans le sens arrière

// on utilise la variable v pour le pilotage PWM

}

void tourneDroite(int v){

Serial.print("Je tourne a droite");

affiche(v);

//fonction de rotation à droite : un moteur dans un sens, l'autre dans le sens opposé

// on utilise la variable v pour le pilotage PWM

}

void tourneGauche(int v){

Serial.print("Je tourne a gauche");

affiche(v);

//fonction de rotation à gauche : un moteur dans un sens, l'autre dans le sens opposé

// on utilise la variable v pour le pilotage PWM

}

void affiche(int v){

//fonction complémentaire d'affichage

if (v==AFOND){

Serial.println(" a fond !");

}

else{

Serial.println(" prudemment...");

}

}

Ce code ne contient pas les commandes pour les moteurs, je vous laisse le faire https://openclassrooms.com/bundles/common/images/smiley/hihi.png.

J'attire tout de même votre attention sur le début du programme. Vous y trouvez la directive (car ce n'est pas une fonction) #define . C'est une directive qui permet de fixer des noms de constantes. Il faut penser à bien écrire les constantes en majuscule pour un meilleur repérage dans le code. On l'utilise comme suit :

#define NOMDECONSTANTE VALEUR

Lors de la compilation, à chaque fois que l'IDE trouvera NOMDECONSTANTE dans le code, il le remplacera par la valeur. Du coup, pas d'utilisation de mémoire dynamique ! Vous rencontrerez souvent cette directive dans les programmes (et ce, dans plusieurs langages !). N'hésitez pas à l'utiliser.

Plusieurs points à noter pour  #define :

* Si dans une de vos variables vous utilisez NOMDELACONSTANTE du coup il sera remplacé par la valeur. exemple : vous faites #define VA 3  puis plus loin vous déclarez la variable  int laVALEUR=0;  alors lors de la compilation, votre variable deviendra : la3LEUR  ce qui ne voudra rien dire et pourra gêner le programme dans certains cas.
* #define est une directive, donc pas de ";" à la fin de la ligne. De même, pas de signe "=" entre le nom et la valeur.
* Arduino conseille de préférer la déclaration de constante en utilisant le mot-clé  const (voir [cours d'initiation](https://openclassrooms.com/courses/programmez-vos-premiers-montages-avec-arduino/utilisez-les-constantes-les-variables-les-conditions-et-le-moniteur-serie#/id/r-3355268)).

Si vous utilisez un capteur infrarouge, il faudra modifier la lecture de la distance (car c'est un pin analogique) et faire les tests de mesure pour faire correspondre infos du CAN et distance.

Bon, vous avez maintenant un robot qui évite les obstacles sans les toucher. Que pourrait-il faire d'autre ?

**Je suis une ligne noire sur le sol**

Ha ben oui ! En voilà une bonne idée !

(Non Lukas, le titre ne veut pas dire qu'il faut s'habiller en noir et s'aplatir par terre, il s'agit du verbe suivre !)

Comment notre robot peut-il réaliser ce tour de force ? Et bien rappelez-vous que la couleur noire réfléchit très mal la lumière.

Ben on peut utiliser une photodiode infrarouge alors ?

(Oui Cunégonde, mais la prochaine fois, laissez-moi le temps de le dire !)

En effet, notre problème de tout à l'heure devient une solution. En utilisant notre capteur de distance infrarouge et en le dirigeant vers le sol, nous obtiendrons une mesure très faible lorsque l'infrarouge rebondira sur du noir, et une mesure plus forte sur d'autres couleurs. Pour une efficacité totale, la couleur blanche est conseillée.

Comment va fonctionner un tel programme ?

Tout d'abord, ça dépend du matériel de l'épaisseur de la ligne.

**Deux capteurs, ligne épaisse (plus de 5 cm)**

C'est le cas le plus simple. Vos deux capteurs sont au dessus de la ligne. Ils renvoient chacun une valeur faible. Si en avançant, le robot s'éloigne de la ligne, l'un des deux capteurs va le remarquer.

Donc si une valeur forte est renvoyée par le capteur de droite, c'est qu'on quitte la ligne par la droite, on doit donc tourner à gauche. À l'inverse, une valeur forte renvoyée par le capteur de gauche signifie qu'il faut aller à droite.

On ajuste donc le mouvement dans la direction opposée du capteur qui détecte la sortie.

Si les deux capteurs envoient en même temps une valeur forte, on vient de sortir de la ligne en allant tout droit. Il faut faire demi-tour et chercher le passage.

**Deux capteurs, ligne fine (de 5mm à 1 cm)**

C'est déjà plus compliqué. Si la ligne est fine, il faut que les capteurs se situent de part et d'autre de celle-ci.

On teste cette fois-ci la chute de valeur (donc survol de la ligne noire).

Si le capteur de gauche "voit" la ligne noire, c'est qu'on se décale vers la droite et inversement, si le capteur de droite "voit" la ligne noire, c'est qu'on se décale à gauche. Donc on ajuste dans la direction du capteur qui repère la ligne.

Si les deux capteurs "voient" la ligne en même temps, c'est qu'on est à une intersection. Il faudra donc trouver un moyen de se retrouver avec une seule ligne entre les deux capteurs.

Il est important dans cette configuration, de faire vérifier au robot qu'il a bien une ligne noire entre ses deux capteurs (avec une rotation vers la droite, puis la gauche), sinon il pourrait quitter la ligne sans s'en apercevoir et aller batifoler n'importe où... ;)

**Un capteur, une ligne de taille minimum (au moins 2 cm)**

Là c'est plus difficile. Ça ne peut fonctionner qu'en contrôlant qu'on est sur la ligne.

Tant que le capteur reçoit une valeur basse (il voit la ligne) tout est ok.

S'il reçoit une valeur haute, c'est qu'il vient de quitter la ligne. Le problème c'est qu'on ne sait pas de quel côté. Il faut dont stopper et faire un balayage droite/gauche pour la retrouver et le remettre en route.

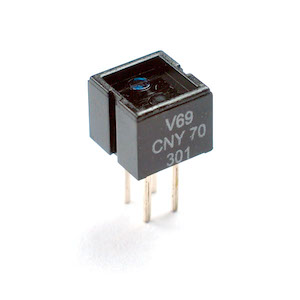
Le mouvement du robot sera moins fluide, plus saccadé. Mais il réussira tout de même sa mission !

**Les contraintes pour la construction**

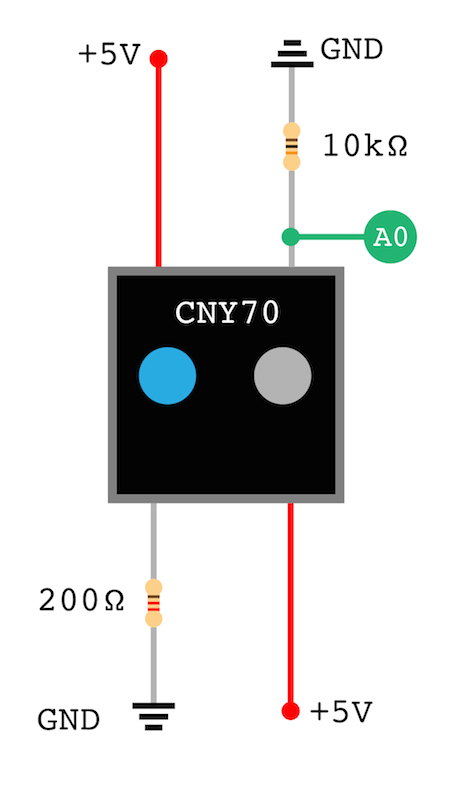
Pour que les données reçues soient claires (valeur haute hors ligne et valeur basse sur la ligne), il faut que le capteur soit situé à bonne distance du sol. C'est un réglage important à faire. Trop proche et la lumière infrarouge ne rebondira pas vers le capteur. Trop éloigné et le capteur donnera des variations trop faibles pour une analyse correcte. L'idéal c'est de concevoir un montage avec un capteur facilement réglable en hauteur.

L'autre point découle du premier. Le capteur étant proche du sol, le robot ne pourra pas se permettre d'évoluer dans un environnement chaotique. Bref, il faut que la zone soit plate et sans obstacles.

J'en profite pour vous montrer un nouveau composant qui est très efficace pour le suivi de ligne noire : le CNY70.

Le CNY70. Émetteur/récepteur infrarouge (http://5hertz.com)

Il a l'air gros comme ça, mais il est tout petit (moins d'1 cm de côté) et il ne coûte par cher (1 à 3€). On le trouve facilement dans les magasins d'électronique ou sur des sites de vente en ligne (il suffit de taper sa référence dans un moteur de recherche). Voici comment le connecter à votre Arduino :

Connexion du CNY70 à l'Arduino

Le repérage des pattes est simple du fait de la différence de couleur.

Il faut le placer assez près du sol pour de bonnes mesures, mais vu le prix, ça vaut le coup de se faire un robot double capteur !

**En résumé**

Nous venons de voir trois missions données à un robot :

* Éviter un obstacle après être entré en contact avec (grâce aux micro-rupteurs) ;
* Éviter un obstacle avant la collision (en utilisant un capteur infrarouge ou à ultrasons) ;
* Suivre une ligne noire sur le sol (grâce aux capteurs infrarouges).

Sommes-nous à la fin des missions ?

Bien sûr que non, il est possible de réaliser bien des choses avec des robots mobiles suffisamment équipés en capteurs. D'ailleurs rien ne vous empêche de cumuler plusieurs sortes de capteurs !

Le plus compliqué reste les réglages et la programmation. Mais si vous avez un objectif clair, c'est bien plus simple.

Voici quelques exemples de missions à réaliser :

* Le robot ne tombe pas de la table sur laquelle il se trouve (difficile).
* Le robot se dirige vers une source de lumière (assez facile avec une [photo-résistance](https://openclassrooms.com/courses/programmez-vos-premiers-montages-avec-arduino/les-capteurs-electroniques#/id/r-3309342), vue dans le cours d'initiation).
* Le robot prend et déplace des objets (il faut lui prévoir une sorte de pince et un moyen de détecter qu'il peut prendre un objet. Il faut aussi qu'il différencie mur et objets à prendre ;)).
* Le robot se dirige vers une source de lumière en restant sur une ligne noire et en évitant des obstacles sans les toucher (réalisable avec vos connaissances, mais nécessite une grosse organisation du programme !).
* Le robot "rebondit sur une ligne noire" (pratique pour délimiter des zones d'action ou des labyrinthes).
* Le robot cherche une ligne noire et la suit (pratique pour un retour à la base).

Je vous laisse inventer la suite !

Bien, nous avons fait le tour de ce que je souhaitais aborder concernant l'utilisation des capteurs avec un robot mobile. Nous allons maintenant apprendre à utiliser notre premier shield : un shield moteur !

**Utilisez un shield moteur (1/2)**

Vous avez appris jusqu'à maintenant à utiliser votre carte Arduino avec des composants électroniques connectés sur une breadboard.

Vous allez apprendre dans ce chapitre à utiliser un shield.

Un shield est une carte qui vient se positionner directement sur votre Arduino. Il vous permet d'utiliser certaines fonctionnalités qui lui sont propres, tout en vous laissant la possibilité de continuer à utiliser les pins libres. Il est possible d'empiler plusieurs shields sur un seul Arduino (s'ils sont compatibles entre eux bien sûr) ! Vous en verrez les avantages, en particulier en termes d'ergonomie et de simplicité de programmation.

Le shield que nous allons utiliser est un shield moteur, c'est-à-dire qu'il permet de piloter des moteurs.

Il existe déjà la L293D pour ça non ?

(Que ferait-on sans vous Cunégonde ?!) En effet, et vous allez justement comprendre les avantages et inconvénients d'un shield par rapport à un circuit intégré L293D.

Pour suivre complètement ce chapitre, il vous faut donc :

* Le [shield moteur d’Adafruit](https://www.adafruit.com/product/1438) ;
* Des [pinheaders](https://www.adafruit.com/products/85) pour mieux connecter votre shield (facultatif) ;
* Du matériel pour soudure à l’étain (fer à souder et... étain  ).

Mais je ne vais pas vous faire patienter plus longtemps ! Voyons de quoi il s'agit !

**Un shield pour Arduino**

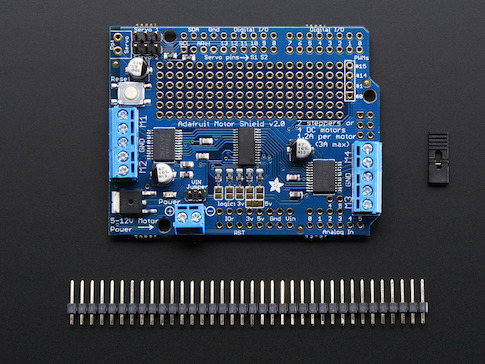
Pour contrôler des moteurs, vous avez vu dans le cours d'initiation que l'on peut utiliser des transistors bipolaires (ou des MOSFET) en concevant un [pont en H](https://openclassrooms.com/courses/programmez-vos-premiers-montages-avec-arduino/le-moteur-a-courant-continu-partie-2-le-pont-en-h-et-les-circuits-integres). Ça faisait pas mal de connexions et ça prenait de la place. Puis vous avez découvert la puce L293D, qui propose 2 ponts en H (4 demi-ponts) le tout dans un petit boîtier noir avec 16 pattes. Ça vous libérait de pas mal de fils et vous n'aviez qu'à vous concentrer sur la connexion de la puce et des moteurs.

Et bien il existe dans le monde de l'Arduino ce qu'on appelle des **Shields**. Shield en anglais signifie bouclier (et oui Lukas, d'où l'acronyme du groupe des héros de Marvel™).

Ok, mais pourquoi l'Arduino aurait-il besoin d'un bouclier ?

Il faut plutôt entendre le mot shield comme une **couche hardware supplémentaire qui vient compléter la carte Arduino**. En effet, les pattes du shield vont venir se connecter exactement et complètement sur la carte Arduino, en la recouvrant. Le shield est aussi conçu pour vous permettre de continuer à avoir accès aux entrées et sorties de l'Arduino. Vous profitez donc des fonctionnalités du shield et de celles de l'Arduino. Bien sûr, certains pins de l'Arduino vont être réservés pour le shield (donc pas utilisables...).

Voici une image de la bête que je vous propose d'adopter :

Adafruit Motor/Stepper/Servo Shield pour Arduino v2 Kit - v2.3 (Adafruit.com)

Il s'agit du "Adafruit motor shield v2.3", le shield moteur fabriqué par Adafruit. C'est un choix personnel pour ce cours. Il existe plusieurs shields moteurs dans le commerce. Alors pourquoi celui-ci ? Pour plusieurs raisons :

* Tout d'abord parce que la société Adafruit, représentée par LadyAda (ce n'est pas une blague https://openclassrooms.com/bundles/common/images/smiley/heureux.png) est très impliquée dans le monde de l'Arduino. Certes pour des raisons commerciales, mais aussi pour permettre au concepts de se développer, et aux utilisateurs/clients de pouvoir utiliser du matériel fiable à un prix convenable.
* Le coût de ce shield est très correct si l'on compare avec d'autres du commerce (en particulier vis-à-vis des fonctionnalités). Il est accessible sur commande aux alentours de 20$. Les délais de livraison sont respectés (en tout cas en ce qui concerne mes commandes). En revanche il faudra vous acquitter d'une taxe de douane lors de la livraison du produit (loi française). Donc comptez plutôt 25€ pour le recevoir chez vous.
* Ce shield contient 4 ponts en H. Il peut donc piloter jusqu'à 4 moteurs dans les deux sens (ou 8 moteurs dans un seul sens). Il permet le pilotage de steppers (moteurs pas-à-pas) dont je ne parlerai pas dans ce cours. Il prévoit aussi des connexions pour deux servo-moteurs.
* Les connexions des moteurs se font grâce à des dominos (bleus clairs sur l'image) qui permettent de glisser le fil et de faire contact en vissant. J'allais dire pas de soudure, mais en fait nous verrons que si. Il propose aussi la connexion d'une alimentation externe (entre 4,5 et 13V) pour les moteurs.
* On peut enficher jusqu'à 32 shields l'un sur l'autre, ce qui permettrait d'utiliser 128 moteurs dans les deux sens !
* Le design et l'ergonomie de l'ensemble est très bien pensé, il est même ajouté une zone avec des trous si l'on veut ajouter des composants au shield.
* Adafruit a développé une bibliothèque d'instructions spéciales pour l'utilisation de ce shield, ce qui est très appréciable ! Nous l'utiliserons d'ailleurs dans le chapitre suivant.

Ça en fait des bonnes raisons ! C'est vrai, mais pour être honnête, je vais parler aussi des inconvénients :

* Une fois reçu, le kit nécessite de réaliser des soudures ! L'acquisition d'un fer à souder demande un investissement supplémentaire d'environ 20€. Mais sachez que vous vous en servirez sûrement bien des fois si la passion de la robotique vous prend. Souder des pattes sur un shield reste une opération assez facile, mais attention aux brûlures pour les maladroits.
* Si l'on veut pouvoir empiler plusieurs shields, il ne faut pas utiliser les headers (pattes) fournis avec le kit, mais en acquérir d'autres (pattes longues et trous) qui peuvent être commandés sur le même site.
* Ce kit Adafruit est fabriqué aux US. Vous aurez donc une taxe de douane à payer à la réception de votre commande en plus des frais de port (le montant de cette taxe dépend de ce que vous avez commandé, je n'ai pas de tarif spécifique pour le shield seul).

Il est tout à fait possible de trouver ce shield sur des sites de commerce français.  Un peu plus cher certes, sûrement à cause des taxes, mais peut-être plus simple pour les non-anglophones (sur [Amazon](https://openclassrooms.com/courses/3290206/parts/3293146/amazon.fr) par exemple).

Mais une question me brûle les lèvres... on parle de connecter 4 moteurs CC, si on compte 2 connexions par moteur plus les 4 contrôles PWM, plus le pilotage des deux servos, on en est à 17 pins utilisés !!! Ça nous prend tout !

(Quel bouillonnement dans ce cerveau Cunégontesque !) En effet le calcul est bon ! Mais la ruse de ce shield (comme plusieurs autres d'ailleurs), c'est qu'il n'utilise que 2 pins analogiques pour piloter les moteurs, et 2 pins numériques pour les servos, ce qui vous laisse encore bien des possibilités !

Comment est-ce possible ?

Tout simplement parce que la carte ne contient non pas deux L293D, mais un circuit intégré qui va piloter un quadruple pont en H. Arduino va communiquer avec ce circuit grâce à un protocole de communication (c'est-à-dire en utilisant un code de discussion précis) pour donner les ordres nécessaires au circuit intégré. Ce protocole s'appelle I2C, et il ne nécessite du coup que deux pins analogiques : le pin 5 et le pin 6.

Le bus de données I2C (Inter-Integrated-Circuit) est un protocole créé initialement par Philips pour des applications domestiques. Il est devenu une des normes de communication entre un microprocesseur et différents circuits intégrés. L'Arduino est donc à même de communiquer en utilisant ce bus.

Là où la société Adafruit se distingue des autres constructeurs, c'est qu'elle propose une bibliothèque complète pour utiliser son shield, "Adafruit motor", en code libre (donc modifiable) et qui va gérer la communication entre l'Arduino et le circuit intégré. Vous allez donc apprendre à télécharger et utiliser cette bibliothèque !

Mais avant de programmer notre shield, voyons comment souder les headers (les petites pattes en métal qui vont permettre la connexion sur l'Arduino).

**Un point sur la soudure**

Pour souder, il vous faut un fer à souder.  Ça se trouve partout (magasins de bricolage, d'électronique, de modélisme, sur Internet..) et ce n'est pas excessif, vu l'utilisation que vous en ferez dans votre carrière de passionné du robot. Si vous ne vous sentez pas d'en acheter un ou de faire les soudures vous-même, je suis sûr que vous avez dans votre entourage un bricoleur qui peut vous dépanner. Au pire, en allant dans un magasin d'électronique, je pense qu'il vous feront les soudures gratuitement (si vous demandez gentiment) ou avec un petit billet (si votre tête ne leur revient pas ;) ).

(Pour vous Lukas, je pense qu'il serait plus sage de vous mettre à la soudure...)

Je ne vais pas vous faire un cours magistral sur la soudure et le matériel à utiliser et tout. Je vais juste attirer votre attention sur deux points : la sécu et la bonne soudure.

**Un point sécu !**

Le fer à souder permet de faire fondre de l'étain afin qu'il fasse contact entre le circuit et le composant. La température de fusion de l'étain est de 231,9 °C. Dire que c'est chaud est un euphémisme.

Voici donc quelques règles de sécurité à l'adresse des inconscients :

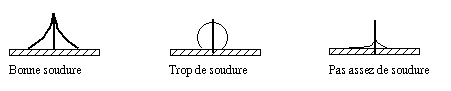
* Ne laissez jamais un fer à souder branché si vous ne l'utilisez pas.
* Reposez toujours le fer sur son socle entre chaque utilisation (on ne le pose pas sur la table en bois du salon).
* Fixez bien le socle sur le plan de travail (ben oui, si il tombe, il ne sert à rien).
* Travaillez dans un environnement sans risque (pas d'enfant proche, ou alors calmes, pas d'essence, de gaz, ou autre produits inflammables...).
* Ne laissez jamais un enfant souder seul (mais accompagné d'un adulte, c'est complètement possible ! Le fer n'est dangereux que mal utilisé).
* Ne vérifiez pas que le fer est chaud avec votre doigt ou toute autre partie de votre corps.
* Pour les Italiens et toute personne qui parle en faisant des gestes, posez le fer sur son socle avant de vous exprimer.

Tout ceci peut paraître évident, mais un petit rappel fait moins mal qu'une brûlure...

**La bonne soudure**

Une bonne soudure est une soudure qui fait le lien électrique correct entre le composant et le circuit. Une soudure qui bouge n'est pas une bonne soudure. Je vais donc vous faire la différence entre une bonne soudure et une mauvaise soudure (oui ce sketch est inconnu).

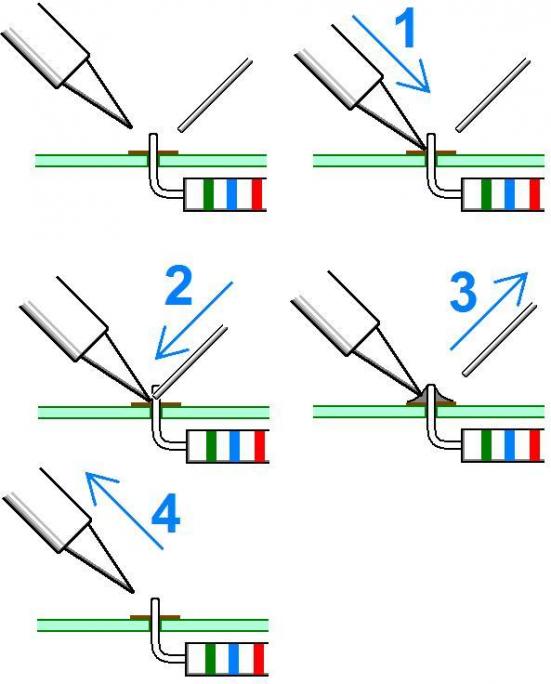
Vous trouverez sur Internet, ou avec les conseils de vos amis spécialistes, bien des explications sur la façon de souder. J'ajoute juste un petit visuel qui permet de garder en tête l'image d'une bonne soudure :

Une bonne soudure (http://meteosat.pessac.free.fr)

Ça a l'air simple comme ça, mais obtenir la forme chapiteau (à gauche) n'est pas si évident au début ! On obtient souvent la forme boule (au centre) qui n'est pas une soudure viable.

L'image "trop de soudure" correspond surtout à une chaleur trop faible de la plaque par rapport à la soudure. Mais retenez que si vous formez une petite bille, ce n'est pas bon.

L'enchaînement des gestes est assez simple (si votre composant et votre plaque sont stables), là encore une explication en images :

Les bons gestes pour souder à l'étain (http://www.astuces-pratiques.fr)

Attention, vous remarquerez une petite erreur dans l’étape 4 de l’image : il manque la représentation du petit chapiteau que l’on voit à l’étape 3.

**Soudez les pattes**

Lorsque vous recevez votre shield, les dominos est les pattes ne sont pas soudées. Je ne vais pas refaire ici ce qui a déjà été très bien fait par le constructeur (toujours le petit plus Adafruit). Je vous invite donc à consulter leur [page d’instructions](https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/install-headers)  pour souder les composants du shield. Même si vous n'est pas à l'aise en anglais, les images de cette page sont largement suffisantes pour tout placer correctement.

Petite information complémentaire tout de même. Comme je l'ai dit précédemment, si vous soudez les pattes fournies, vous obtiendrez un shield qui ne permet pas de recevoir un autre shield ou d'utiliser les pins de l'Arduino. C’est là qu’entrent en jeux les pin-headers spéciaux (longues pattes et trous) que je vous ai conseillé de commander en début de cours. Voici le [lien](https://www.adafruit.com/products/85). Ils vont vous permettre par la suite de pouvoir connecter d'autres shields ou tout simplement d'utiliser vos pins libres.

À propos des commandes sur Internet et en particulier à l'étranger :

Si vous savez ce que vous voulez et que vous savez où le commander, je vous conseille fortement de grouper vos commandes ! En effet, les frais de port et les taxes ajoutent au coût du produit.

C'est dommage de devoir payer 5€ de frais pour 4 LED à 0,1€ l'unité.

Faites donc le tour de tout ce dont vous avez besoin (jumpers, pin-header, résistances, diodes, moteurs, servo...) et faites un tir groupé. ;)

Une fois votre shield monté et prêt, voyons comment l'utiliser !

**Connectez le shield et les moteurs à la carte Arduino**

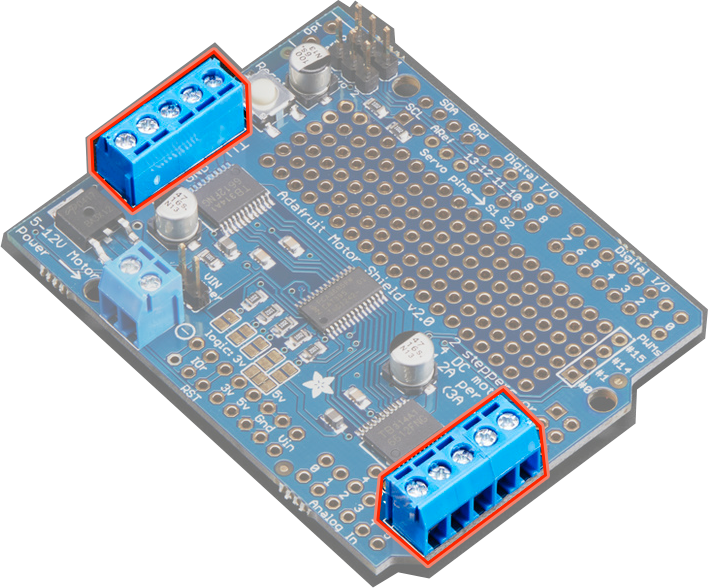
Alors pour connecter le shield, c'est pas bien difficile, et je pense que vous n'avez pas eu besoin d'une explication pour réussir : il suffit d'enficher le shield sur l'Arduino. Et c'est tout ! Bien sûr ça marche si vous avez tout soudé au bon endroit...

(Non Lukas, ce n'est pas je m'enfiche, tu t'enfiches, il s'enfiche... "Enficher" c'est faire correspondre les bonnes pattes du shield sur les bons trous de l'Arduino !)

Inutile d'appuyer comme une brute, il suffit que les contacts se fassent correctement. Si vous enfoncez trop votre shield, ce sera plus difficile de l'enlever par la suite.

La connexion des moteurs au shield est, elle aussi, d'une simplicité enfantine !

Vous avez deux dominos de chaque côté du shield :

Les dominos de connexion des moteurs

Ils servent à brancher vos moteurs. Sur chaque domino, on peut connecter 2 moteurs pour les faire tourner dans les deux sens. Pour chaque moteur on va utiliser deux trous à vis. Ils sont repérés par un marquage sur la carte : M1, M2, M3 et M4.

Si vous utilisez les moteurs dans les deux sens (comme c’est notre cas ici), vous n’avez rien à connecter sur le GND. Le GND sert pour les moteurs pas-à-pas et les moteurs utilisés dans un seul sens.

Vous pouvez choisir les connexions qui vous conviennent le mieux, n'en n'utiliser qu'une ou les quatre, vous êtes libres.

Si vous utilisez une source d'énergie externe (et c'est conseillé), il suffit de la brancher sur le domino restant en respectant les bornes + et -.

Pensez bien dans ce cas à enlever le jumper qui est placé juste à côté, si vous l'aviez précédemment mis.

Une fois tout ceci connecté, nous pouvons passer à la suite !

**Utilisez un shield moteur (2/2)**

Poursuivons notre découverte du shield moteur...

**Téléchargez et installez la bibliothèque Adafruit motor**

Pour que ce shield fonctionne, il lui faut une série d'instructions contenues dans une bibliothèque (library) qui vous est fournie par Adafruit.

Pour que tout soit bien plus simple, je vous conseille de mettre à jour l'IDE de l'Arduino sur votre ordinateur. En effet, il y a une fonctionnalité très efficace pour télécharger et installer les bibliothèques.

Je n'expliquerai pas une installation "à la main" vu que l'IDE propose une façon simple et efficace de le faire, et que la mise à jour de l'IDE est toujours fortement conseillée. Je sais je ne suis pas **que** sympa.

Voici la procédure à suivre :

1. Ouvrez votre IDE.
2. Allez dans la barre de menu : Croquis -> Include Library -> Manage libraries...
3. Dans la fenêtre du "Library manager" qui s'ouvre, tapez "Adafruit motor" dans le champ de recherche. (Si vous ne tapez que Adafruit, vous aurez un aperçu de l'apport des bibliothèques Adafruit disponibles pour la communauté.)
4. Choisissez la version qui correspond à votre shield (c'est noté sur le shield, la récente est v2.3) et la version de la bibliothèque (le manager propose la dernière automatiquement).
5. Cliquez sur "install".
6. Attendez que l'installation se fasse et fermez tout (même l'IDE).
7. Ouvrez l'IDE à nouveau, et si tout se passe bien, dans Fichier -> Exemples vous devriez voir apparaître "Adafruit Motor Shields V2 Library" (en fonction de votre version).
8. Ayé, c'est fait ! Il ne nous reste plus qu’à découvrir les fonctionnalités de code liées au shield moteur disponibles dans cette bibliothèque..

**Faites tourner un moteur**

Pour la suite de ce chapitre, je pars du principe que :

* Vous avez connecté votre shield moteur sur votre carte Arduino.
* Vous avez la version v2 du shield moteur Adafruit.
* Vous avez téléchargé la bibliothèque qui va bien(voir section précédente).
* Vous avez connecté votre moteur sur la sortie M1 du shield.

SI vous alimentez l'Arduino et le shield par une source différente (ce que je vous conseille), n'oubliez pas d'enlever le jumper derrière le domino.

Adafruit indique que le shield risque de ne pas fonctionner avec des moteurs dont la tension de fonctionnement est inférieure à 5V. De mon côté, je n'ai pas encore eu de problème à ce niveau. Le risque de mon point de vue est plutôt de griller le moteur.

Adafruit indique aussi qu'on ne peut pas alimenter le shield avec une batterie de 9V (pile carré). Là encore, ça ne pose aucun problème et mes moteurs tournent tous. Mais la pile se décharge vite.

Je le signale tout de même, car du coup je ne suis pas dans les clous https://openclassrooms.com/bundles/common/images/smiley/triste.pngmais comme c'est mon matériel... à vous de décider pour le vôtre.

Commencez par lancer un nouveau programme (sketch) dans l’IDE Arduino.

**Déclarez vos ressources**

La première chose à faire dans votre programme, c’est d’inclure les bibliothèques pour qu'il puisse les compiler ensuite :

* La bibliothèque "wire" qui permet la communication I2C avec le shield. Celle-ci est déjà présente par défaut dans l'IDE, donc pas besoin de la télécharger au préalable;
* La bibliothèque "Adafruit\_MotorShield" qui gère le shield moteur (que vous avez téléchargé dans la section précédente).

Pour inclure des bibliothèques, vous connaissez la procédure :

#include <Wire.h>

#include <Adafruit\_MotorShield.h>

On doit ensuite créer un objet shield pour que l'Arduino puisse discuter avec. Cette objet a une adresse générique I2C (c'est-à-dire un code qui permet de savoir que c'est avec lui que l'Arduino discute). Cette adresse est0x60. Il est possible de modifier cette adresse avec des soudures sur le shield. Ce n'est pas l'objet de ce chapitre.

Le principe du changement d'adresse se fait en créant un nombre sur 5 bits qui s'ajoute à l'adresse 0X60. Le codage dépend des soudures réalisées sur les zones près du power.

J'appelle mon objet shield "monShield", libre à vous de lui donner le nom que vous voulez.

Adafruit\_MotorShield monShield = Adafruit\_MotorShield();

Ensuite, on crée un objet moteur qu'on rattache à l'objet shield, en indiquant la sortie utilisée de cette façon :

Adafruit\_DCMotor \*monMoteur = monShield.getMotor(1);

Le signe \* situé devant le nom de l'objet moteur est important ! C'est un petit code qui veut dire qu'on ne travaille pas avec la variable, mais avec son adresse en mémoire. C'est ce qu'on appelle un pointeur. Je ne vous expliquerai pas ici leur utilité, mais une lecture du cours sur le langage C vous en dira bien plus long.

J'ai décidé d'appeler le pointeur "monMoteur", là encore vous pouvez choisir un autre nom.

Nous en avons fini avec les déclarations liées au shield. Nous n'utilisons ici qu'un seul moteur. Nous verrons plus loin comment utiliser 2 moteurs.

**Instructions dans le blocsetup()**

Dans le  setup()i il faut démarrer la communication avec le shield avec la fonction :

monShield.begin();

Bien, à partir de là nous pouvons utiliser notre moteur.

Les moteurs sont programmables en vitesse et en sens de rotation.

Pour paramétrer la vitesse, on utilise la fonction  setSpeed(valeur)  où  valeur  est un entier entre 0 (arrêt) et 255 (vitesse maximum). Mais comme on utilise un pointeur (et non un objet), on ne va pas utiliser le  .  pour appeler la fonction liée à l'objet, mais la flèche  ->  (tiret et signe supérieur). Du coup pour définir la vitesse au maximum pour notre moteur, je vais coder :

monMoteur->setSpeed(255);

Maintenant que la vitesse de rotation est fixée, vous pouvez démarrer votre moteur. Pour cela, il faut utiliser la fonction  run(). Elle attend un paramètre qui peut prendre trois valeurs :

* FORWARD qui signifie en avant ;
* BACKWARD qui signifie en arrière ;
* RELEASE qui signifie stop.

On les écrit en majuscules car ce sont des constantes fixées par la bibliothèque du shield.

Donc pour lancer le moteur en avant, j'écris :

monMoteur->run(FORWARD);

Là encore on utilise le signe  ->.

Normalement, lorsque vous utilisez les instructions d'une bibliothèque, la coloration syntaxique du code met les mots-clés d'une autre couleur (souvent en orange), ce qui vous permet de repérer d'éventuelles erreur de frappe.

Voici donc le programme qui fait tourner le moteur dans un sens :

#include <Wire.h> //bibliothèque pour la communication I2C

#include <Adafruit\_MotorShield.h> //bibliothèque pour le shield

Adafruit\_MotorShield monShield = Adafruit\_MotorShield(); //création de l'objet shield

Adafruit\_DCMotor \*monMoteur = monShield.getMotor(1); //création de l'objet moteur par pointeur et repérage du numéro

void setup() {

monShield.begin(); //On lance la communication avec le shield

monMoteur->setSpeed(255); //On définit la vitesse de rotation

monMoteur->run(FORWARD); //On fait tourner le moteur

}

void loop() {

}

Vous pouvez téléverser et tester le programme. Le moteur devrait tourner.

Le sens avant (FORWARD) ou arrière (BACKWARD) dépend de vos attentes. Si le moteur tourne vers ce que vous décidez être l'arrière, ne modifiez pas le code, inversez juste les connexions du moteur. Il devrait alors tourner dans le bon sens.

J'attire votre attention sur la LED verte "power" située sur le shield. Elle indique que le shield est correctement alimenté en tension.

**Faites tourner plusieurs moteurs**

Alors le code qui permet de faire tourner plusieurs moteurs est en fait très simple. Au lieu de déclarer un moteur, on en déclare deux !

Comme d'habitude, les noms que vous choisirez vous permettront de mieux lire votre code !

Voici donc un code qui vous permet de démarrer deux moteurs (moteurGauchee et  moteurDroite) ensemble pendant une seconde, puis qui les arrête pendant une seconde, répété à l'infini :

#include <Wire.h> //bibliothèque pour la communication I2C

#include <Adafruit\_MotorShield.h> //bibliothèque pour le shield

Adafruit\_MotorShield monShield = Adafruit\_MotorShield(); //création de l'objet shield

Adafruit\_DCMotor \*moteurGauche = monShield.getMotor(1); //création de l'objet moteurGauche par pointeur et repérage du numéro

Adafruit\_DCMotor \*moteurDroite = monShield.getMotor(2); //création de l'objet moteurDroite par pointeur et repérage du numéro

void setup() {

monShield.begin(); //On lance la communication avec le shield

moteurGauche->setSpeed(255); //On définit la vitesse de rotation

moteurDroite->setSpeed(255); //des deux moteurs

}

void loop() {

moteurGauche->run(FORWARD); //On fait tourner les moteurs

moteurDroite->run(FORWARD); //ensemble dans le même sens

delay(1000); // pendant 1 seconde

moteurGauche->run(RELEASE); //Puis on les arrête

moteurDroite->run(RELEASE); //tous les deux

delay(1000); // pendant 1 seconde

}

Cette fois-ci, j'ai mis des instructions dans la boucle  loop()  pour que le programme se répète.

Si les deux moteurs ne tournent pas dans le même sens (qui correspond pour vous à l'avant), là encore, il suffit d'inverser les connexions du moteur qui ne tourne pas comme attendu, ou des deux si besoin. ;)

Il est bien sûr possible de créer des fonctions qui démarrent ou arrêtent les moteurs !

Tiens d'ailleurs, que diriez-vous d'un petit exercice ?

(Oui, bien sûr Lukas, on peut commencer par une petite pause aussi...)

**TP : Écrivez les fonctions de déplacement de votre robot**

Dans le chapitre précédent, vous avez conçu des programmes qui permettaient de faire réagir vos robots en fonction de l'environnement, grâce aux capteurs.

Nos programmes n'étaient pas complets, puisqu'il manquait le contenu des fonctions :

* arret(),
* avance(int v),
* recule(int v),
* tourneDroite(int v), et
* tourneGauche(int v).

où  v  représente la valeur pour la vitesse du robot.

Je vous propose donc de créer ces fonctions en utilisant la bibliothèque du shield, ce qui vous permettra de les mettre dans le programme final de votre robot.

Le programme doit définir les bonnes fonctions, et les appeler successivement dans la  loop()  (oui Lukas, l'une après l'autre) et attendre 1 seconde entre chaque appel.

Vous avez largement tout ce qu'il vous faut pour réaliser cet exercice, alors au boulot !

(Cunégonde, puisque vous avez déjà fini, allez donc aider Lukas à retrouver son Arduino...)

**TP : Correction**

Le code là encore est simplifié par l'utilisation du shield et de sa bibliothèque d'instructions. Vous remarquerez d'ailleurs que c'est aussi le cas de votre montage, il y a beaucoup moins de fils ! C'est justement tout l'avantage des shields.

Voici donc la correction de l'exercice attendu (ou la correction attendue de l'exercice...) :

#include <Wire.h> //bibliothèque pour la communication I2C

#include <Adafruit\_MotorShield.h> //bibliothèque pour le shield

Adafruit\_MotorShield monShield = Adafruit\_MotorShield(); //création de l'objet shield

Adafruit\_DCMotor \*moteurGauche = monShield.getMotor(1); //création de l'objet moteurGauche par pointeur et repérage du numéro

Adafruit\_DCMotor \*moteurDroite = monShield.getMotor(2); //création de l'objet moteurDroite par pointeur et repérage du numéro

void setup() {

monShield.begin(); //On lance la communication avec le shield

moteurGauche->setSpeed(255); //On définit la vitesse de rotation

moteurDroite->setSpeed(255); //des deux moteurs

}

void loop() {

arret();

delay(1000);

avance(255);

delay(1000);

recule(255);

delay(1000);

tourneDroite(255);

delay(1000);

tourneGauche(255);

delay(1000);

}

void arret(){

//fonction d'arrêt des deux moteurs

moteurGauche->run(RELEASE);

moteurDroite->run(RELEASE);

}

void defVitesse(int v){

moteurGauche->setSpeed(v); //on redéfinit la vitesse

moteurDroite->setSpeed(v); //des deux moteurs

}

void avance(int v){

//fonction de marche avant

defVitesse(v); //appel de la fonction pour définir la vitesse

moteurGauche->run(FORWARD);

moteurDroite->run(FORWARD);

}

void recule(int v){

//fonction de marche arrière

defVitesse(v);

moteurGauche->run(BACKWARD);

moteurDroite->run(BACKWARD);

}

void tourneDroite(int v){

//fonction pour tourner à droite sur place

defVitesse(v);

moteurGauche->run(FORWARD);

moteurDroite->run(BACKWARD);

}

void tourneGauche(int v){

//fonction pour tourner à gauche sur place

defVitesse(v);

moteurGauche->run(BACKWARD);

moteurDroite->run(FORWARD);

}

Vous remarquez que j'indique qu'il s'agit de tourner sur place. En effet, vous pouvez être plus subtile dans la rotation :

* Si vous arrêtez une roue et laissez l'autre en marche, la rotation se fera autour le la roue arrêtée.
* Si vous définissez une roue à une vitesse et l'autre à une vitesse moindre, votre robot tournera sur une courbe plutôt que sur place ou autour de sa roue.
* Vous pouvez aussi vous contenter de garder une roue à une vitesse constante, et d'augmenter ou de diminuer la vitesse de l'autre, ce qui vous permet des trajectoires plus fines.

Et bien voilà une affaire qui roule !

**Les connexions pour servo-moteurs**

Vous trouverez sur le shield, deux connexions pour servo-moteurs. Elle ne sont pas pilotées par la bibiothèque du shield. En fait c'est juste que ces connexions sont reliées au pins 9 (servo 2) et 10 (servo 1) de l'Arduino. Il vous suffit de brancher correctement les fils du servo-moteur (le *gnd* vers le haut et le fil de commande vers les pins numériques) pour que votre servo soit correctement connecté au +5V et *gnd* de l'Arduino. Ce n'est donc pas l'alimentation des moteurs qui sera utilisée.

Pour piloter des servos, on utilise la bibliothèque "Servo" proposée par l'IDE.

Vous trouverez les explications liées aux [servo-moteurs](https://openclassrooms.com/courses/programmez-vos-premiers-montages-avec-arduino/donnez-du-mouvement-a-vos-montages-avec-un-servo-moteur) (fonctionnement et programmation) dans mon cours d'initiation.

**En résumé**

Je pense que vous avez bien avancé sur votre robot mobile !

En effet dans ce chapitre vous avez :

* Appris à réaliser quelques soudures ;
* Connecté le shield moteur à l'Arduino ;
* Téléchargé et utilisé les bibliothèques pour utiliser ce shield ;
* Appris à piloter un moteur, puis deux moteurs ;
* Appris à simplifier votre programme pour piloter un moteur à l'aide de fonctions.

Vous pourriez aussi ajouter à votre robot des LED qui s'allument en fonction de ses mouvements pour améliorer le côté visuel.

Dans le chapitre suivant, je vous propose de lui ajouter des sons !

**Générez des sons**

Maintenant que votre robot (grâce à la carte Arduino) sait se déplacer et éviter les obstacles, nous allons voir comment lui faire produire des sons.

Je vous arrête tout de suite, votre robot n’aura pas accès à une synthèse vocale perfectionnée, mais il pourra produire des sons simples et de petites mélodies.

Vous allez donc comprendre le principe électrique de production du son, et voir le matériel nécessaire pour le faire avec votre Arduino.

Pour suivre ce chapitre, il vous faut donc un haut parleur piezzo ou buzzer ou petit haut-parleur.

Vous verrez aussi la différence entre son et note, et comment l’on peut programmer une mélodie.

Enfin, si vous êtes sages, je vous fournirai une petite mélodie connue ;).

**Créez un son avec un signal carré**

Un petit point sur le son...

**Le son**

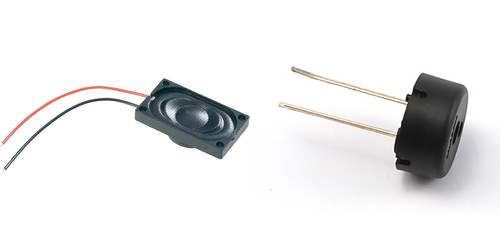
Le son est avant tout une onde. C’est-à-dire, une déformation d’un fluide élastique (l’eau ou l’air par exemple),  qui se propage en s’éloignant de sa source. Notre oreille est capable de percevoir des sons.

Pour vous donner une analogie (oui je suis assez friand de ça), imaginez une petite étendue d’eau plate et calme. Vous tapez à la surface de cette eau. La déformation que vous créez à l’endroit du choc, va se répandre en ondes concentriques (qui ont le même centre) sur toute la surface de l’eau. Si vous placez un bouchon avec un capteur de hauteur, vous réussirez à repérer la hauteur de l’onde (sa puissance) et la longueur de l’onde (sa fréquence)  
Et bien c’est la même chose pour le son.

Une corde de guitare qui vibre, va transmettre sa vibration à l’air. Cette vibration va se propager dans l’air jusqu’à vos oreilles (notre tympan est comme le bouchon mesureur) qui vont recevoir la vibration. Notre cerveau analyse le tout pour nous donner une traduction mentale de cette vibration.

Mais comment produire un son avec du matériel électronique ?

Il nous faut un matériel qui va pouvoir vibrer sur commande ! Pour cela on utilise un piezzo ou un petit haut-parleur.

Un petit haut-parleur et un Piezzo

L’un comme l’autre contient une partie vibrante. Quand je dis vibrante, c’est qu’elle peut passer dans une position puis dans l’autre, très vite. On pourrait parler de position haute et basse.

En fait la position haute ou basse du piezzo ou du buzzer est mécanique et fonctionne sur le principe de l'électro-aimant (vu dans le cous d'initiation à l'Arduino). Lorsqu'il est parcouru par un courant l'électro-aimant positionne une membrane à l'état bas (par exemple) et lorsqu'il n'est plus parcouru de courant, il se relâche et la membrane se remet à son état normal haut (par exemple). C'est la rapidité de ce mouvement qui crée la vibration.

Comme les trains d’impulsion ?

(Dans le mille Cunégonde !) En fait, c’est justement grâce à un train d’impulsions que l’on va créer un son. Si l’on envoie un signal carré à un piezzo (ou à un haut parleur), il va vibrer en correspondance avec ce signal et produire un son.

Nous l’avons vu dans le [cours d'initiation](https://openclassrooms.com/courses/programmez-vos-premiers-montages-avec-arduino/les-capteurs-electroniques#/id/r-3248171) : pour qu’un son soit audible, il faut que la vibration se situe entre 20 Hz et 20000 Hz. Nous verrons que pour l’Arduino, nous n’avons pas besoin de telles extrémités de fréquences.

Voyons comment produire un signal carré.

**Le signal carré**

Nous l’avons dit, il s’agit de placer le piezzo à l’état haut puis à l’état bas en fonction d’une fréquence donnée.

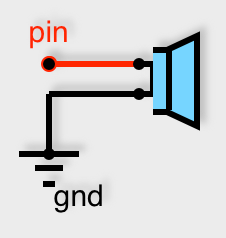
Prenons la fréquence de 440 Hz. Ça voudrait dire que le piezzo doit faire un aller-retour 440 fois par seconde. Pour calculer la période, c’est-à-dire le temps d’un aller-retour, on utilise la formule suivante :

*fr*é*quence*(*Hz*)=1/*p*é*riode*(*s*)

donc 1/440=0,002272 secondes soit 2272 microsecondes. Ceci pour un aller-retour. Donc un aller ou un retour donne 2272/2=1136 microsecondes.

Le signal carré doit être régulier, donc autant de temps en haut qu’en bas (contrairement à d’autres trains d’impulsions).

Voici comment se connecte un buzzer ou un piezzo sur l'Arduino :

Connexion d'un piezzo ou buzzer sur l'Arduino

Certains montages proposent d'ajouter une résistance avant d'aller au ground. Cela diminue le volume sonore, mais protège votre Arduino. Personnellement, je ne le fais pas.

Nous allons donc écrire un programme qui place le pin connecté au piezzo, à l’état haut pendant 1136 microsecondes et à l’état bas pendant 1136 microsecondes :

void setup() {

pinMode(3,OUTPUT); //on prépare le pin 3 en mode sortie

}

void loop() {

digitalWrite(3,0); // état bas

delayMicroseconds(1136); //on attend 1136 milli-secondes

digitalWrite(3,1); // état haut

delayMicroseconds(1136); // on attend 1136 millisecondes

}

Vous devriez entendre un son sortir du piezzo ou du haut-parleur.

Si vous mesurez ce son avec un [accordeur](https://fr.wikipedia.org/wiki/Accordeur_(outil)) (matériel pour musicien qui aime jouer juste), il vous indiquera que votre son vibre à une fréquence de 434,3 Hz. Nous ne sommes pas loin des 440 Hz. La différence s'explique par le temps que chaque fonction met à s'exécuter. En effet, les fonctions  digitalWrite(),  delayMicroseconds() et  loop() prennent un tout petit peu de temps pour s'exécuter, elles ralentissent donc légèrement le processus.

Ce n'est pas grave en soi dans notre cas, la justesse du son n'est pas notre priorité pour le moment.

En musique, plus la fréquence est élévée (donc la période basse), plus le son est aigu. Donc si on fait varier le délai d'attente on obtient des sons plus ou moins aigus.

Voici un programme qui crée un petit son de laser (à l'ancienne) :

int p=10; // on prépare une variable pour la période

void setup() {

pinMode(3,OUTPUT); //on prépare le pin en mode sortie

}

void loop() {

p++; // on incémente p à chaque boucle de loop();

if (p>500){ // teste la limite de p

p=10; // on le remet à 10

delay(500); // on attend un peu

}

digitalWrite(3,0); // état bas

delayMicroseconds(p); //on attend p milli-secondes

digitalWrite(3,1); // état haut

delayMicroseconds(p); // on attend p millisecondes

}

Le son part d'une fréquence haute (car la période est petite, de 10 micro-secondes) vers une fréquence basse (de période plus grande, de 500 micro-secondes), et enchaîne très rapidement ces sons, d'où l'effet du son du laser (comme on peut l'entendre dans des films ou des jeux).

Petite précision scientifique :

* Un laser ne fait du bruit que dans les films, en réalité la lumière ne produit pas de son.
* Lorsqu'un vaisseau explose dans l'espace, en réalité il ne produit aucun son. En effet, il faut un fluide (air, eau...) pour qu'un son se propage, or l'espace est vide ! On pourrait vous crier au ras de l'oreille (si le casque n'était pas obligatoire) que vous n'entendriez rien. ;)

*Exercice* : Je vous propose d'essayer par vous-mêmes de créer le son du saut genre [Donkey-Kong](https://fr.wikipedia.org/wiki/Donkey_Kong_(personnage)). Il vous suffit de faire l'inverse, c'est à dire de partir d'une fréquence basse vers une fréquence haute... (le son produit une sorte de "boiiiing" numérique).  
Retrouvez la solution dans la partie Annexe du cours !

C'est très bien tout ça, mais ça ne nous dit pas comment faire des mélodies !

En effet, pour la mélodie, il faut connaître un peu le principe des tons (non Lukas, rien à voir avec la pêche !).

**Le ton**

Chaque note que joue un musicien est à une fréquence précise. En effet, tout à l'heure,  en produisant un son à 440 Hz, nous avons joué un LA !

La gamme européenne comprend 12 notes. Chacune d'elle est à une fréquence précise. Voici un exemple de tableau des fréquences :

| Note | Fréquence |
| --- | --- |
| Do | 65 Hz |
| Do# ou Réb | 69 Hz |
| Ré | 74 Hz |
| Ré# ou Mib | 78 Hz |
| Mi | 83 Hz |
| Fa | 87 Hz |
| Fa# ou Solb | 93 Hz |
| Sol | 98 Hz |
| Sol# ou Lab | 104 Hz |
| La | 110 Hz |
| La# ou Sib | 117 Hz |
| Si | 123 Hz |

Le # signifie dièse (et pas hashtag ) et le b signifie bémol. Ce sont des altérations des notes de la gamme de base (Do, Ré, Mi, Fa, Sol, La, Si). Le dièse augmente la fréquence de la note et le bémol la diminue. Ainsi un La # est situé en fréquence entre le La et le Si (ce qui est vérifié par le tableau).

Mais il n'y a pas que 12 notes sur un piano ?!

(Je vois que vous êtes musicienne Cunégonde...). Chaque touche du clavier d'un piano est en effet une note, et il y en a bien plus que 12 ! En fait une fois qu'on a passé toutes les notes, on recommence avec le do, mais sur une fréquence plus élevée (plus aiguë).

Chaque groupe de note (de Do à Si) est appelé une octave. Et il y a un truc facile à retenir, c'est que pour passer une note d'une octave à l'octave supérieure, il suffit de multiplier sa fréquence par 2.

Prenons l'exemple du la :

* Octave 0 : La à 110 Hz
* Octave 1 : La à 220 Hz
* Octave 2 : La à 440 Hz (on retrouve bien notre La de tout à l'heure !)

Donc pour jouer une note dans une octave supérieure, il nous suffit d'appliquer la formule suivante :

fréquence=(fréquence à l'octave 1) x (2 puissance octave)

Pour trouver notre La 440 Hz on pouvait faire 110x2^2=440.

En réalité, on dit que le La à 440 Hz est un La 4, c'est-à-dire 4ème octave par rapport aux possibilités musicales de production et d'écoute de son. Ici je pars d'un La à 110 Hz pour fixer l'octave 0, ce qui est largement suffisant pour nos sons.

Retenons donc deux choses :

* Les fréquences des notes de la gamme sont fixes.
* Pour passer d'une octave à la supérieure, on multiplie la fréquence par 2.

Mais produire une mélodie ne repose pas que sur la fréquence...

**Le rythme**

Si vous chantonnez au clair de la lune, vous verrez que le son des 4 première syllabes est dit régulièrement (au/clair/de/la) et que les syllabes de lune sont dites plus longuement (luuuuuuu/neeeeeeee).

Tout simplement parce qu'on tient plus longtemps la note. Si l'on veut faire une mélodie, il nous faut donc pouvoir tenir la fréquence de la note un certain temps, puis passer à la note suivante.

Une mélodie est donc une succession de notes, qui ont une durée (un rythme) et une octave. En programmation,  on s'imagine assez vite un tableau du genre :

int note[3]={ton, rythme, octave};

qui nous permettrait de stocker une note. Et pour en stocker plusieurs, et bien on ferait un tableau à deux dimensions :

int notes[nbMaxNotes][3];

N'oublions pas que pour déclarer un tableau, il faut en connaître les limites (ou au moins en fixer les limites).

Alors pour que notre mélodie ne soit pas totalement illisible, nous allons créer des constantes qui reprennent la valeur des fréquences. On peut utiliser la directive #define TON frequence  ou déclarer avec la forme proposée par l'IDE Arduino :

const char TON=frequence;

Je choisis la seconde solution et je vous donne le code. Il est à placer au début du programme :

const char DON = 65;

const char DOD = 69;

const char REN = 74;

const char RED = 78;

const char MIN = 83;

const char FAN = 87;

const char FAD = 93;

const char SON = 98;

const char SOD = 104;

const char LAN = 110;

const char LAD = 117;

const char SIN = 123;

J'ai choisi de noter DON le DO Normal pour avoir le même nombre de lettre partout. Donc pour comprendre, on utilise les deux premières lettres de la note et on ajoute N pour normal ou D pour dièse.

Sachez qu'il existe une banque de toutes les fréquences sur 8 octaves sur [cette page de la documentation Arduino](https://www.arduino.cc/en/Tutorial/ToneMelody). Les notes sont indiquées en dénominations Anglaise (A=La, B=SI, C=Do, D=ré, E=Mi, F=Fa, G=Sol). C'est en fait, de mon point de vue, inutile de créer toutes ces constantes, du fait de la possibilité de faire le calcul de chaque note par octave si on a les fréquences de départ. À vous de juger ce que vous préférez.

Pour la durée, nous pouvons choisir un code simple :

Si on imagine que le temps (c'est ce qui est mesuré quand vos tapez du pied en écoutant de la musique) est le code 2, on a :

* code 1 : la note dure la moitié d'un temps (croche en musique)
* code 2 : la note dure 1 temps (noire en musique) donc 2 croches
* code 4 : la note dure 2 temps (blanche en musique) donc 4 croches
* code 8 : la note dure 4 temps (ronde en musique) donc 8 croches

Du coup, voici comment stocker "Au clair de la lune" dans un tableau :

int auClair[11][3]={

DON, 2, 2,

DON, 2, 2,

DON, 2, 2,

REN, 2, 2,

MIN, 4, 2,

REN, 4, 2,

DON, 2, 2,

MIN, 2, 2,

REN, 2, 2,

REN, 2, 2,

DON, 8, 2

}

Voilà, notre mélodie est stockée (enfin le début seulement).

Cool, et on en fait quoi maintenant ?

Et bien, il nous faut créer un interpréteur musical. C'est à dire un programme qui va transformer ce tableau en musique !

Pour créer cet interpréteur, il nous faut prévoir le moment auquel la note doit changer !

Alors pour réaliser ceci il vous faut pouvoir gérer le temps ! En effet, comment allez-vous gérer le passage d'une note à l'autre si vous ne savez pas si un temps suffisant est écoulé ?

Alors j'en profite (car c'est surtout fait pour vous apprendre des choses ce cours) pour vous montrer une méthode très pratique pour gérer le temps : les fonctions  millis()  et micros().

**Gérez le temps qui passe**

Notre carte Arduino est cadencée à 16 MHz, soit 16 000 000 d'actions machine effectuées par seconde. Il existe dans l'Arduino, des variables spéciales qui comptent le temps qui passe. Nous pouvons avoir accès à ces données de plusieurs façons : soit par les fonctions toutes prêtes, soit en allant chercher dans l'utilisation de registres (mais vous n'en êtes pas là).

Nous allons utiliser deux fonctions toutes prêtes :

* La fonction  millis(), qui ne prend pas d'argument et qui nous retourne le temps écoulé en millisecondes depuis le dernier "reset" de l'Arduino
* La fonction  micros(), qui ne prend pas d'argument et qui nous retourne le temps écoulé en micro-seconde depuis le dernier "reset" de l'Arduino.

Si vous deviez agir à la microseconde près, la fonction micro() serait un peu trop lente, mais pour la musique, c'est largement suffisant !

Alors l'idée est simple : on compte le temps qui passe. Si le temps écoulé dépasse la durée (le rythme) de la note, on fait changer la note, sinon, on joue la note.

Voici un morceau de code qui affiche un point sur la console toutes les secondes :

unsigned long tempsDebut; // variable pour stocker le temps de départ

unsigned long tempsActuel;// variable pour stocker le le temps qui s'écoule

unsigned long duree;// variable pour stocker la duree d'attente

void setup() {

Serial.begin(9600);// on initialise la communication série

tempsDebut = millis(); // on initialise le temps de départ sur le temps Arduino

duree = 1000; // on initialise la durée d'attente à 1 seconde

}

void loop() {

tempsActuel = millis(); //on stocke le temps Arduino

if (tempsActuel - tempsDebut >= duree) { // on teste si la durée est dépassée

Serial.print("."); // on écrit un point sur la console

tempsDebut = tempsActuel; //on remet le temps de départ au temps Arduino

}

}

Mais quel est l'intérêt, vu qu'on peut utiliser la fonctiondelay()  ?!

(Vous me décevez Cunégonde, je pensais que vous devineriez !)

La fonction  delay()  met le programme en pause, donc aucune autre action n'est possible ! En revanche, avec le programme ci-dessus, on peut demander à l'Arduino de faire autre chose en attendant que le moment d'afficher le point arrive. Le test que nous réalisons avec le comptage du temps nous permet donc de libérer le programme tant que le moment n'est pas encore arrivé.

Ce principe est très utile et très utilisé en programmation. Il existe même des registres spéciaux de l'Arduino (les timers) qui permettent de réaliser ce genre de comptage et d'actionner les fonctions au bon moment (avec des interruptions).

Avec ce code vous avez créé une sorte de timer.

Et pour être encore plus efficaces, voyons la fonction de son créée pour l'Arduino !

**La fonction tone()**

Nous avons vu tout à l'heure comment générer un signal carré qui produit un son avec un haut-parleur.

Notre programme fonctionnait bien, certes, mais il lui manquait une chose importante : si je joue un son, je ne peux pas faire autre chose !

Et bien, grâce aux timers dont nous avons parlé à l'instant, les ingénieurs de l'Arduino on créé la fonction tone().

Cette fonction permet de générer un signal carré correspondant à la fréquence voulue. Mais comme ils utilisent un timer (donc une fonction interne à l'Arduino qui compte le temps qui passe) pour faire vibrer à la bonne fréquence, entre chaque état (haut ou bas) du piezzo, l'Arduino est libre d'exécuter une autre tâche.

Donc si on exécute la fonction  tone() dès le début du programme (dans le  setup()  par exemple) nous pourrions ensuite réaliser toutes les actions voulues, avec un son permanent dans nos haut-parleurs (je parle bien d'un son unique, et non de mélodie).

Alors comment utiliser cette fonction ;) ? Et bien là encore, c'est très simple :

* On doit mettre le pin concerné en mode OUTPUT.
* On appelle ensuite la fonction soit en utilisant tone(pin, fréquence)  ou  tone(pin, fréquence, durée). La valeur de durée est en millisecondes.
* Si on ne met pas de durée, le son continue à l'infini. On peut l'arrêter en faisant appel à la fonction noTone(pin) , qui va stopper le son sur le pin concerné.

Plusieurs points de mise en garde :

* L'utilisation de la fonction  tone() interfère sur les possibilités PWM des ports 3 et 11 sur l'Arduino UNO. Tout simplement parce qu'elle utilise le timer qui est réservé à ces deux pins pour le PWM.
* Si vous voulez jouer une note sur un autre pin, il faudra d'abord stopper la note jouée sur le pin actuel. On ne peut donc pas jouer plusieurs notes à la fois avec cette fonction sur l'Arduino UNO.
* La fonction  tone() ne permet pas de jouer des fréquences inférieures à 31 Hz. Là encore, il s'agit du timer qui est codé comme un  int, donc avec des limites.

Je vous laisse modifier les programmes "laser" et  "saut" de tout à l'heure en utilisant la fonction  tone(). En testant un peu les possibles, vous pourrez rapidement avoir une bibliothèque de bruitages à ajouter à votre robot !

Et si nous créions notre interpréteur ?

**L'interpréteur de mélodie**

"Voici venu le temps des rires et des chants" (Casimir)

Résumons ce que doit contenir notre interpréteur :

* La liste des fréquences de base ;
* Un tableau qui stocke la mélodie ;
* Un moyen d'attendre un moment défini entre chaque note ;
* Une fonction qui renvoie la bonne fréquence en fonction de la fréquence de base et de l'octave.

Nous allons décider que l'attente entre chaque note est d'une demi-seconde.

Voici tout d'abord le programme "simple" d'interpréteur. Il ne permet pas de réaliser d'autres actions car nous utilisons la fonction delay()  :

// les constantes des fréquences de base

const char DON = 65;

const char DOD = 69;

const char REN = 74;

const char RED = 78;

const char MIN = 83;

const char FAN = 87;

const char FAD = 93;

const char SON = 98;

const char SOD = 104;

const char LAN = 110;

const char LAD = 117;

const char SIN = 123;

//Le tableau pour la mélodie

char auClair[11][3]={

DON, 2, 2,

DON, 2, 2,

DON, 2, 2,

REN, 2, 2,

MIN, 4, 2,

REN, 4, 2,

DON, 2, 2,

MIN, 2, 2,

REN, 2, 2,

REN, 2, 2,

DON, 8, 2

};

int dureeBase=500; //on fixe la durée de basse à 500 millisecondes

void setup(){

pinMode(3,OUTPUT);//on met le pin 3 en mode OUTPUT

}

void loop(){

for (int n=0;n<11;n++){ // boucle de lecture du tableau

char note = auClair[n][0]; // on récupère la fréquence de base dans le tableau

char octave= auClair[n][2]; // on récupère l'octave

int frequence=freqNote(note,octave); //on calcule la bonne fréquence

int duree=dureeBase\*auClair[n][1];// on multiplie la duree de base par la valeur de duree du tableau

tone (3,frequence); //on joue la note

delay(duree);

noTone(3);//on arrete la note

delay(10);// petite attente pour avoir l'impression d'attaquer la note

}

}

// fonction de calcul de la fréquence en fonction de l'octave

int freqNote(int fb,int oc){

return fb\*pow(2,oc);

}

Vous noterez au passage l'utilisation de la fonction mathématique :  pow(nombre1, nombre2)  qui renvoie  nombre1  à la puissance  nombre2.

Vous pouvez modifier le nombre mis dansdureeBase , vous verrez que la musique accélère ou ralentit.

Il est tout à fait possible de récupérer la valeur d'un potentiomètre pour régler la valeur de  dureeBase  donc la vitesse d'exécution de la mélodie. https://openclassrooms.com/bundles/common/images/smiley/clin.png

Et voici maintenant la même chose mais en mode multitâches :

// les constantes des fréquences de base

const char DON = 65;

const char DOD = 69;

const char REN = 74;

const char RED = 78;

const char MIN = 83;

const char FAN = 87;

const char FAD = 93;

const char SON = 98;

const char SOD = 104;

const char LAN = 110;

const char LAD = 117;

const char SIN = 123;

//Le tableau pour la mélodie

char auClair[11][3]={

DON, 2, 2,

DON, 2, 2,

DON, 2, 2,

REN, 2, 2,

MIN, 4, 2,

REN, 4, 2,

DON, 2, 2,

MIN, 2, 2,

REN, 2, 2,

REN, 2, 2,

DON, 8, 2

};

int dureeBase=500; //on fixe la durée de basse à 500 millisecondes

unsigned long tempsDep; // variable pour le temps de départ

unsigned long tempsAct; // variable pour le temps actuel

int duree; //variable pour la durée d'attente de la note en cours

int n=0; // position dans le tableau de mélodie

void setup(){

pinMode(3,OUTPUT);//on met le pin 3 en mode OUTPUT

tempsDep=millis(); // on initialise le temps de départ au temps Arduino

duree=0; //on initialise l'attente à 0

}

void loop(){

tempsAct=millis(); // on récupère le temps Arduino

if (tempsAct-tempsDep>=duree){ // on regarde si le temps est écoulé

noTone(3); // on stoppe le son

delay(10); // délay pour l'attaque du son

joueNote(auClair[n][0],auClair[n][2]); // on appelle la fonction qui joue la bonne note

duree=dureeBase\*auClair[n][1]; // on fixe la duree d'attente

tempsDep=tempsAct; //on initialise le temps de départ

n++; // on incrémente la position dans le tableau

if (n>10) // on teste si on dépasse la fin du tableau

n=0; // on revient au début du tableau

}

// on peut placer ici du code à excécuter en attendant

// il faut bien-sûr ne pas utiliser la fonction delay() ;)

}

// fonction de calcul de la fréquence en fonction de l'octave

void joueNote(int nt,int oc){

tone(3,nt\*pow(2,oc)); //on joue la note à la bonne fréquence

}

Il est tout à fait possible de placer l'ensemble du bloc lié à la mélodie dans une fonction pour plus de clarté dans la boucle  loop().

Voici une petite dédicace à notre ami Lukas. Le programme joue une mélodie (connue ;) ). On peut tout à fait exécuter d'autres actions dans la  loop() pendant ce temps à condition qu'elles n'utilisent pas de  delay().

/\*

Petit clin d'oeil mr Lukas

Nanomaître 2015

\*/

// constantes de fréquences

#define DON 33

#define REB 35

#define REN 37

#define MIB 39

#define MIN 41

#define FAN 44

#define SOB 46

#define SON 49

#define LAB 52

#define LAN 55

#define SIB 58

#define SIN 62

// tableau pour la mélodie

int part[50][3] = {

DON, 2, 100,

DON, 2, 100,

DON, 2, 100,

LAB, 1, 75,

MIB, 2, 25,

DON, 2, 100,

LAB, 1, 75,

MIB, 2, 25,

DON, 2, 200,

SON, 2, 100,

SON, 2, 100,

SON, 2, 100,

LAB, 2, 75,

MIB, 2, 25,

SIN, 1, 100,

LAB, 1, 75,

MIB, 2, 25,

DON, 2, 200,

DON, 3, 100,

DON, 2, 100,

DON, 3, 25,

SIN, 2, 25,

DON, 3, 25,

0, 0, 75,

SIN, 2, 50,

SIB, 2, 100,

SIB, 1, 100,

SON, 2, 25,

SOB, 2, 25,

SON, 2, 25,

0, 0, 75,

SOB, 2, 50,

FAN, 2, 100,

SON, 1, 100,

SIB, 1, 100,

SON, 1, 75,

MIB, 2, 25,

DON, 2, 100,

LAB, 1, 75,

MIB, 2, 25,

DON, 2, 200,

-1

};

int pinSon = 3; // pin de connection du haut-parleur

int tempo = 120; // variable du tempo

int duree = 0; // variable de durée de note

unsigned long tempsDep; // variable de temps de départ

int p = 0; // variable de position dans le tableau de mélodie

void setup() {

pinMode(pinSon,OUTPUT);

tempsDep = millis(); // initialisation du temps de départ

}

void loop() {

joue(); // appel de la fonction pour jouer la mélodie

}

//fonction de lecture de la mélodie

void joue() {

unsigned long tempsAct = millis();

if (tempsAct - tempsDep >= duree) {

if (part[p][0] != -1) { // test de fin de tableau

noTone(pinSon);

delay(10); // délai pour l'attaque

// la fréquence est calculée en fonction des fréquences de base

// et de l'octave définit dans le tableau

int frequence = part[p][0] \* pow(2, part[p][1] + 1);

// la durée de la note est calculée comme en musique

duree = 1000 / (tempo / 60) \* (float(part[p][2]) / 100);

if (frequence > 0) {

tone (pinSon, frequence);

}

p++; //incrémentation de la position dans le tableau

}

else {

noTone(pinSon);

p=0;// retour au début du tableau

duree=1000;// attente avant répétition

}

tempsDep=tempsAct;

}

}

J'ai utilisé des  #define pour les constantes, et j'ai pris les valeurs de fréquences plus petites.

Du fait de la complexité de la mélodie, j'ai codé le tempo et les durées de notes sur 100.

J'espère que les fans apprécieront  ;).

**En résumé**

Vous avez appris dans ce chapitre :

* À produire des sons avec votre Arduino (bruitages et mélodies) soit en produisant un signal carré, soit en utilisant la fonction tone()
* À gérer le défilement du temps en utilisant les compteurs  millis()  et  micros() .
* À créer un interpréteur musical à l'aide de tableaux de quelques fonctions.

Vous avez maintenant, à l'issue de cette partie, largement de quoi créer un robot mobile, autonome et musical !!!

Si nous passions à des modes de communication avancés ?